PCT WORLD INTELLECTL
Inter

WO 9603715A1

INTERNATIONAL APPLICATION PUBLISHED

(51) International Patent Classification 6 : **G06T 1/60, 15/00, G06F 12/06**	A1	(11) International Publication Number: **WO 96/03715** (43) International Publication Date: 8 February 1996 (08.02.96)

(21) International Application Number: PCT/AU95/00445

(22) International Filing Date: 21 July 1995 (21.07.95)

(30) Priority Data:
PM 7013 22 July 1994 (22.07.94) AU

(71) Applicant *(for all designated States except US)*: MONASH UNIVERSITY [AU/AU]; Wellington Road, Clayton, VIC 3168 (AU).

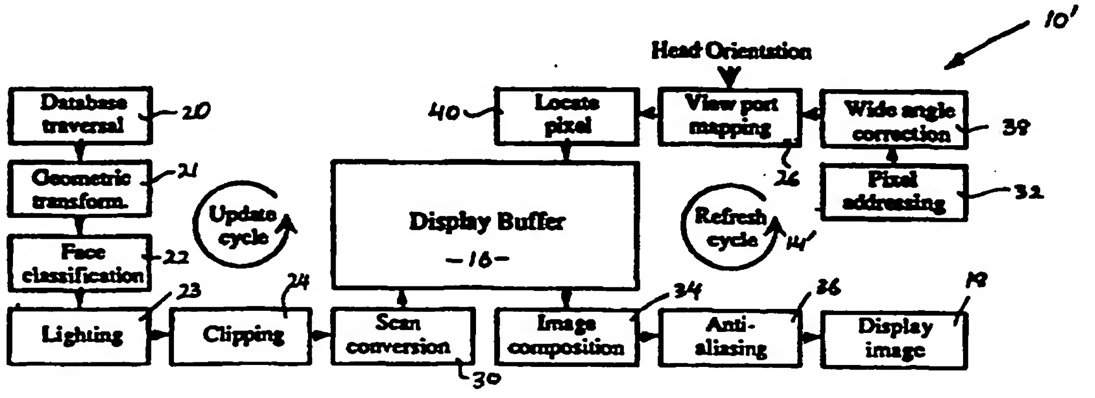
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: POSE, Ronald, David [AU/AU]; 66 Orrong Crescent, Caulfield North, VIC 3161 (AU). REGAN, Matthew, James, Patrick [AU/AU]; 25 Chapel Street, Glen Waverley, VIC 3150 (AU).

(74) Agents: WEBBER, David, Brian et al.; Davies Collison Cave, 1 Little Collins Street, Melbourne, VIC 3000 (AU).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, US, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD, SZ, UG).

Published
With international search report.

(54) Title: A GRAPHICAL DISPLAY SYSTEM



(57) Abstract

A graphical display system which renders images on surfaces encapsulating a user using a prioritised rendering method. The rendered images for the viewport are accessed using an address recalculation pipeline. The image data is accessed by determining a rotational orientation for the image to be displayed and addressing the stored image data on the basis of the determined rotational orientation. The pixel data is displayed by computing a first vector indicative of a pixel location on the display with respect to a viewpoint for the display, computing a second vector indicative of a spatial orientation of the display, combining said first and second vectors into a resultant vector, determining an intersection point of the resultant vector with a substantially closed notional surface when projected from a fixed interior projection point, and mapping said intersection point onto stored image data so as to address image data for display at said pixel location. The system includes at least one graphic renderer for generating image data corresponding to stored object display information for objects to be displayed by the system, a plurality of display memories for storing image data generated by the at least one renderer, and a selection stage for determining which of the plurality of display memories to store image data for a particular object on the basis of a relative position of that object to the user. The display memories are periodically updated with image data from the renderer, and the period between updates is different between different display memories.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LU	Luxembourg	TD	Chad
CS	Czechoslovakia	LV	Latvia	TG	Togo
CZ	Czech Republic	MC	Monaco	TJ	Tajikistan
DE	Germany	MD	Republic of Moldova	TT	Trinidad and Tobago
DK	Denmark	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	US	United States of America
FI	Finland	MN	Mongolia	UZ	Uzbekistan
FR	France			VN	Viet Nam
GA	Gabon				

A GRAPHICAL DISPLAY SYSTEM

This invention relates to a graphical display system, such as for displaying signals from a computer apparatus, which can find application in the field of virtual reality systems.

In recent times virtual reality has become a growing area within the field of computer graphics, with many and varied applications such as flight and other training simulations, scientific visualisation as well as the expanding entertainment field. Many systems suffer from inadequate performance resulting in a lack of realism and even psychological side effects.

Current research is directed toward developing an integrated visualisation system which creates virtual worlds and represents them through computer graphics. In order to render real world environments as faithfully as possible, enormous processing power is required. Current systems tend to be characterised by lengthy, noticeable delays between user gestures and system responses. Such interactive latency may result in serious physiological side effects such as the induction of motion sickness.

For example, researchers and practitioners have long been grappling with the serious problem of latency in responding to movement of a user in a virtual environment. This problem is most difficult to handle in the visual domain since the sheer quantity of data and processing required to update the view delivered to the user is beyond the capabilities of even high-end computer graphics system. The problem, while simply described as the lag between a user's head position or orientation changing and the updating of the displayed virtual view to reflect that change, has rather severe consequences, and has eluded solution by many practitioners. A particularly disturbing effect is that of motion sickness which occurs when the brain's expectation of orientation and position is not matched by appropriate sensory input, in this case visual input. To put it simply the view displayed is not what the brain expects so disorientation and perhaps motion sickness can occur.

The three major components of a virtual reality display system are a head-mounted display providing images to the user, a transducer to measure the position and orientation of the user's head and finally an image generator. The head-mounted display provides a window to the virtual 3D world. The position and orientation of this window is directly related to the position and orientation of the user's head. Thus when the user's head moves, the view of the 3D world as seen through the head-mounted display changes appropriately.

Head-mounted displays present a computer generated view of a virtual world to a human user. Such displays generally contain two small physical display devices, recently colour Liquid Crystal Display screens have become the dominant choice. Each display device is presented independently to one eye, allowing three dimensional binocular images to be presented to the user.

The other major component within the display device is a set of wide-angle viewing lenses. These lenses warp the image so as to present an image covering most of the user's field of view. This type of lens is of particular interest within this display system because while covering a large field of view is desirable, the more the lenses warp the image, the more computationally expensive the images are to create. This is because the distortions caused by the lenses have to be compensated with expensive computations during image creation.

A head-mounted tracking system is a fundamental component of a virtual reality system. Experimental and commercial head tracking units employ tracking techniques based upon magnetism, mechanics and optics. While all three techniques are valid, there exists a large difference in the application, expense and performance between the techniques.

Magnetic devices such as the Polhemus 3Space Isotrack, tend to be the most prevalent with current systems. The devices consist of a magnetic source and a small detection unit mounted on the user's head containing three mutually perpendicular coils. When a user gesture is made, the relative position and orientation of the detector changes, resulting in orientation information which is digitised and sent to the display system. Although such systems have a limited operating spaces, more recent magnetic trackers operate with as little as 4 milliseconds latency.

The purpose of an image generator within an interactive graphics system is to maintain the illusion of immersion within a virtual world. This is achieved by completely redrawing the virtual scene based upon the position and orientation of the user's head whenever the user's position or orientation changes. The scene must also be redrawn whenever any animation within the scene is to occur. The rate at which the scene is redrawn is referred to as the frame rate. For the illusion of immersion within a virtual world to be maintained frame rates in the order of ten to sixty plus frames per second are required. Increasing the frame rate within an interactive display system is a desirable characteristic.

Completely redrawing a scene is a computationally expensive operation. Realistically convincing scenes require many polygons to be drawn for each frame. While cartoon–like environments may be produced with as few as five hundred polygons, scenes with tens of thousands of polygons are still far from totally realistic.

The total number of polygons per second required for each of the user's eyes within a virtual reality system is the product of the number of polygons required per frame and the total number of frames per second to be displayed. For a fixed polygon rendering rate, the polygons per frame are in direct opposition to the display frame rate, hence a trade–off between the two is often required. As the polygon rendering rate directly affects the scene realism and scene fluidity, many researchers attempt to improve the performance of their systems by focussing mainly upon the polygon rendering engine. Special purpose graphics architectures designed specifically for high levels of user interaction have therefore been contemplated based upon extremely powerful polygon rendering engines.

One approach to building virtual reality systems has been to slow the update rate of the display so as to enable the computer system to cope with the processing load.

This leads to a rather jerky displayed world with latency being at least the time between updates of the display. Experience with motion pictures and with television has shown that the higher the frame rate the better.

An alternative approach is to reduce the processing load by reducing the quality and resolution of the images displayed. While the display update rate is increased and latency somewhat reduced with this approach, a degree of realism in the virtual world must be sacrificed, leaving one with cartoon-like imagery rather than anything approaching photo realism.

The state-of-the-art in conventional virtual reality systems rely on using the fastest available graphics processing and rendering equipment. This tends to be very expensive with the very best systems costing over a million dollars. In effect the problem is tackled indirectly through the use of massive amounts of processing power. Even so it can be observed that the trade-off between latency and image quality still has to be made, and that with reasonably realistic looking scenes the latency is still a significant problem. An extremely successful virtual reality computing system is the PixelPlanes system developed at the University of North Carolina. This uses thousands of processors to give extremely impressive rendering performance and very good latency characteristics, however PixelPlanes is rather expensive technology which most developers could not afford.

In accordance with the present invention there is provided a method for selecting image pixel data for an image to be displayed comprising:

determining a rotational orientation for the image to be displayed; and

addressing stored image pixel data on the basis of the determined rotational orientation.

The invention also provides a method for displaying a pixel on a display means, comprising the steps of:

computing a first vector indicative of a pixel location on said display means with respect to a viewpoint for the display means;

computing a second vector indicative of a spatial orientation of the display means;

combining said first and second vectors into a resultant vector;

determining an intersection point of the resultant vector with a substantially closed notional surface when projected from a fixed interior projection point; and

mapping said intersection point onto stored image data so as to address image data for display at said pixel location.

The invention further provides a method for displaying an image comprising:

storing image data representing images visible from at least substantially any rotational orientation of a fixed viewpoint;

determining a rotational orientation of a viewing means;

selecting a portion of said stored image data on the basis of the determined rotational orientation; and

displaying an image with said viewing means utilising said selected image data.

The invention also provides a method for displaying an image on a viewing means comprising:

storing image data representing images projected onto a surface at least substantially encapsulating a central viewpoint;

selecting a portion of said stored image data on the basis of an orientation of the viewing means; and

displaying an image with said viewing means using the selected image data portion.

The invention further provides a method for displaying an image to a user comprising:

storing image data representing views for at least substantially any rotational orientation of the user;

sensing a rotational orientation of the user;

selecting a portion of said image data on the basis of said sensed rotational orientation; and displaying an image to the user utilising said selected image data.

The invention also provides a method for addressing image pixel data comprising:

generating and storing image pixel data representing a panoramic view encompassing views from a plurality of rotational orientations from a fixed viewpoint;

sensing a rotational orientation of a viewing means; and

addressing pixels of the stored image data on the basis of the sensed rotational orientation and raster scanning signals for the viewing means.

The invention further provides a method for reducing rotational latency in a head–mounted graphical display system, such as for virtual reality applications comprising:

generating and storing digitised image data for reproducing images representing a plurality of orientational views of a scene visible from a fixed viewpoint;

sensing the orientation of a head–mounted display means; and

selecting image pixel data from the digitised image data for display on the head–mounted display means, the image pixel data being selected synchronously with the display thereof and on the basis of said sensed orientation.

The invention also provides a method for generating pixel control signals for a graphical display means comprising:

generating pixel vector data on the basis of a pixel screen location of the display means and rotational orientation data;

addressing stored image data corresponding to said pixel screen location and said rotational orientation data on the basis of said pixel vector data; and

accessing the addressed image data and generating pixel control signals therefrom.

The invention further provides an apparatus for displaying a digitised graphical image comprising:

a storage means for storing image pixel data representing a panoramic view from a viewpoint;

a head–mounted display means including a display device and means for generating an orientation signal representative of the rotational orientation of the display means; and

means for selecting image pixel data from the storage means on the basis of said orientation signal and a pixel scanning signal for the display means, and generating display signals for controlling said display device on the basis of the selected image pixel data.

The invention also provides a graphical simulation system comprising:

a digital memory for storing data representing a panoramic view rendered onto a notational surface at least substantially surrounding a viewpoint;

a display means capable of rotational orientation changes and including means for generating data indicative of the rotational orientation of the display means; and

a display controller for retrieving image data from said memory in accordance with rotational orientation data from the display means so as to control the display means to generate an image representing a view from the viewpoint at the rotational orientation of the display means.

The invention further provides a display controlling apparatus for use with a display apparatus including imaging means for displaying pixilated image data to the eyes of a user and rotational orientation sensing means for sensing the rotational orientation of the user's head comprising:

a display memory buffer for storing image data representing a panoramic image rendered onto a notional surface substantially surrounding a viewpoint; and

a display controller for accessing image data in the display memory buffer in accordance with the sensed rotational orientation of the user's head, and for generating display signals for said imaging means to display an image corresponding to a view of said panoramic image from the sensed rotational orientation at the viewpoint.

The invention also provides a graphical display controller pipeline for generating pixel control signals for a display means comprising:

a first stage for generating pixel vector data on the basis of a pixel screen location of the display means and rotational orientation data;

a memory location conversion stage for addressing image data corresponding to said pixel screen location and rotational orientation data on the basis of said pixel vector data; and

a pixel generation stage for retrieving the addressed image data and generating therefrom pixel control signals for the display means.

The invention further provides a graphical display controller pipeline for generating pixel control signals for a head-mounted display means which includes a display device and rotational orientation sensing means comprising:

a first stage for generating pixel vector data on the basis of a pixel screen location for the display device;

a second stage for generating resultant vector data from the pixel vector data and the sensed rotational orientation of the display means;

a third stage for generating at least one display memory address from the resultant vector data;

a fourth stage for retrieving display pixel data from at least one display memory on the basis of the at least one display memory address; and

a fifth stage for generating pixel control signals for the display device on the basis of said display pixel data.

The invention also provides a method for composing images in a graphic display system comprising:

periodically rendering and storing first image data in a first display memory at a first update rate;

periodically rendering and storing second image data in a second display memory at a second update rate; and

generating output image data on the basis of the first and second image data, wherein said first update rate is greater than said second update rate.

The invention further provides a graphic display system comprising:

at least one graphic rendering means for generating image data corresponding to stored object display information for objects to be displayed by the system;

a plurality of display memories for storing image data generated by the at least one rendering means; and

selection means for determining which of said plurality of display memories to store image data for a particular object on the basis of a characteristic of that object.

The invention also provides an image data processing apparatus comprising:

a rendering engine for generating first and second image data;

first and second storage means for storing image data;

selection means for selecting which of the first and second image data to store in the respective first and second storage means; and

an image composition means for comparing equivalent pixels from said first and second image data and selecting therebetween for generation of output image data,

wherein said rendering engine generates the first and second image data at different average rates.

The invention further provides an image data processing apparatus for generating output image data representing objects comprising a visual display, the apparatus comprising:

at least one rendering engine for generating image data representing objects comprising a visual display;

a plurality of display memories for storing image data generated by the at least one rendering engine, the display memories being, in use, periodically updated with image data with the update period being different as between different display memories;

a selection means for determining which of the plurality of display memories to store image data representing a particular object on the basis of a calculated validity period for that object indicative of relative movement of the object on the visual display; and

image composition means for retrieving image data from the plurality of display memories and generating output image data therefrom according to a comparison of depth field data for the retrieved image data indicative of the relative depths of the objects represented thereby within the visual display.

The invention also provides a method for processing stored image pixel data for display, the stored image pixel data being mappable onto a notional display surface, comprising the steps of:

calculating an intersection point of a viewport vector with said notional display surface;

accessing stored data for a plurality of pixels identified as adjacent the intersection point when mapped to said display surface; and

interpolating the accessed data on the basis of the relative positioning of said intersection point and the adjacent pixel mapping locations to generate an output pixel value.

The invention further provides a display memory architecture wherein adjacent pixel values of a graphic image are stored in separate memory devices so as to be simultaneously accessible.

Preferred embodiments of the present invention are described hereinafter, by way of example only, with reference to the accompanying drawings, wherein:

Figure 1 is a block diagram illustrating latency to head rotations within a conventional virtual reality display system;

Figure 2 is a block diagram illustrating head rotation latency within a virtual reality system according to a preferred embodiment of the invention;

Figure 3 is a block diagram illustrating the structure of a conventional graphical display system;

Figure 4 is a more detailed block diagram of a conventional graphical display system;

Figure 5 is a block diagram of a display system according to a preferred embodiment of the invention;

Figure 6 shows a panoramic image (left hand side) from a display memory and a view from the stored image (right hand side) from a particular orientation;

Figure 7 is a diagram illustrating relative head rotation latencies in a conventional display system and a display system according to the preferred embodiment of the invention;

Figure 8 is a block diagram of an address recalculation pipeline of the preferred embodiment;

Figures 9 and 10 are block diagrams showing additional features of the address recalculation pipeline designed for a spherical encapsulating surface;

Figure 11 shows a construction of logical elements for a matrix multiplication stage of the pipeline;

Figure 12 is a block diagram of a vector conversion stage of the pipeline;

Figure 13 shows images illustrating a wide-angle lens mapping stage of the pipeline;

Figure 14 is a block diagram illustrating an image composition stage of the pipeline;

Figure 15 is a graphical representation for deriving a translational validity period for objects to be rendered in accordance with another preferred embodiment of the invention;

Figure 16 is a graphical representation for deriving a size validity period for objects to be rendered;

Figure 17 is a time line graph showing a relative sequence of display memory updates;

Figure 18 is a graph of Relative Object Rendering Load (RORL) against minimum feature size;

Figure 19 is a bar chart showing rendering loads at various display memory update rates;

Figure 20 is a graph of the percentage of image objects invalidated against head rotation angle for various minimum feature sizes;

Figure 21 is a diagram illustrating a linear interpolation strategy for reducing aliasing;

Figure 22 is a block diagram of a hardware structure for the linear interpolation strategy;

Figure 23 is a block diagram of a display memory map showing segment

In order to deal with the problem of latency and to understand the development of the present invention, it is helpful to forget for a moment the conventional ways of computer graphics and instead investigate the overall virtual environment.

The problem as stated earlier is one of mismatch between what is displayed to the user and what should be displayed according to the virtual world model in which the user is situated. This mismatch usually is due to the latency involved in updating the display to reflect the virtual reality. The easy solution, involving reducing image quality so one can keep up with the virtual world, creates a mismatch of a different kind but still spoils the illusion of virtual reality.

It is helpful to examine the extent to which activity occurs in the virtual world itself, such as some part of the virtual environment changing or moving in some way. While indeed this does occur, it has been found that in general only a small part of the virtual world changes at any time, and typically the virtual world contains many static objects or scenes. Thus it is possible to deal with such activity without recomputing everything, and since the objects involved are generally independent of the user a slight delay in reflecting their changes does not cause the serious problems mentioned above.

The other activity which occurs in the virtual world is more significant, and involves the movement of the user within the virtual environment. Any slight movement of the user's head leads to a change in everything the user sees and thus appears to the user as though the observed environment has moved completely. It can therefore be appreciated that it is relative movement between the user and the objects in the virtual world which is observable to the user, and it is the user's own movement which has the most dramatic effect since it affects the display of even static parts of the virtual world. The various types of user movement in the virtual world and their consequences are discussed below.

Consider firstly a displacing movement (translation) such as the action of walking forward. When translating, objects in the world appear to move relative to the subject. Close objects appear to move more than distant ones. Very distant objects may not appear to move much at all. Thus for translations in a virtual environment the rendering of objects designated as being close to the user will require greater processing than the distant background in which changes can largely be ignored. In typical scenes the majority of the complexity of the scene is in the background.

Another movement relates to changes in the rotational aspect of the user's head. Even a small rotational change immediately causes everything in the observed view to appear to move, even the background and static objects. There is also the added problem that rotations generally occur faster and more frequently than translations. In other words, a person tends to look around and observe the environment by rotating head and eyes at a faster rate and to a greater extent than moving through the environment.

From the above discussion, it would appear that the environment remains relatively static while the user's view of the environment changes quite rapidly and somewhat unpredictably, mainly due to rotation. The point is that the environment itself does not change dramatically and so it may be possible to find a way in which the graphical representation of the displayed environment also stays relatively static for simplified scene computation and rendering.

A model is described herein in which the images of the virtual world are rendered onto a surface or surfaces surrounding the user. It may appear that in so modelling the world a larger processing bottleneck has been created, both in generating the images and rendering them, and in selecting the appropriate view to display. On first examination it may appear that significantly greater rendering overheads such as scan conversion, clipping, etc. are incurred than a conventional system, however this is rarely the case and is only found if the scene has polygons evenly spread out in all three dimensional directions. With a viewport independent display memory one must scan convert all polygons received. This is the worst case scenario for a conventional system but for guaranteed interactive response one must allow for the worst case. Many conventional rendering systems are designed to cope with situation approaching the worst case scenario. The rendering overheads for a conventional system may be reduced if the user is not looking at a complex part of the scene, however as the system has no control over the user's choice of direction for viewing it is fair to assume the user is looking at the most polygonally dense section of the world. The viewport mapping is indeed computationally expensive and thus a hardware circuit is also described directed to performing the mapping operations.

The nature of rotational latency within a traditional virtual reality display system is illustrated in the block diagram of Figure 1. The steps which comprise the process for displaying an image begin with computing the user's head orientation utilising the head tracking apparatus. Once the head orientation has been determined it is then necessary to create the details of the scene which is to be displayed corresponding to that orientation. After creation of the scene the computationally expensive rendering process can commence. Upon completion of the frame drawing process the image is ready to be displayed. When the display process for the current frame has completely finished, the display process for the new frame may commence. It is apparent that in this form of virtual reality display system the rotational latency of the overall system is dependent upon the latency of each stage of the display process. More particularly, the rotational latency is related to the time required to compute and render a new image each time an orientation change occurs.

By redesigning the display controller within a display system it is possible to perform useful computations as a pixel is being drawn. In particular, a new computer graphics display controller architecture which is suited to the requirements of head-mounted virtual reality (VR) display systems is described in detail hereinbelow, which differs from standard display controllers in the nature of the video memory addressing mechanism. Rather than fetching pixels for display from the video memory sequentially, the fetch mechanism bases the pixel fetch location on the pixel's actual screen location (as seen through a wide-angle viewing lens) and the orientation of the user's head. This means the viewport mapping is delayed until after the scene has been rendered. In effect the user's head orientation is considered after the scene is drawn instead of before the scene is drawn, avoiding the rendering bottleneck. As a result, latency to user head rotation is based only upon the nature of the display device and the head tracking system. In comparison to Figure 1, Figure 2 is a block diagram of a new display system which illustrates the causes of rotation latency within the redesigned display controller.

Providing a stereoptic view of the scene complicates the issues involved in head rotation latency because some re-rendering is required. Although many researchers have found that providing low latency is much more desirable than stereoscopy, the display system described herein includes mechanisms for depth buffering so objects close to the user may be re-rendered to provide a stereoptic view of the object without the need for redrawing the entire scene.

When generating a scene for a virtual reality system there appears to be several processes being undertaken. For ease of reference, these processes have been divided roughly into two major categories. The first category is scene animation and encompasses processes such as object movement within the virtual world, user translation within the virtual world and lighting or shadowing changes within the scene. The second process is scene maintenance and refers to updating the display based upon user gestures such as head rotations. A Static Virtual Environment (SVE) is defined an environment in which only scene maintenance is required.

Traditional virtual reality display systems draw no major distinction between scene maintenance and scene animation, often scene maintenance being achieved through scene animation. The display system described herein in connection with various forms of the invention dedicates hardware within the display controller to the task of scene maintenance. The rendering engines may then use all of their resources for animations within the scene and initial scene creation.

Consider the computations required when a user's head undergoes some form of rotation. The new scene presented to the user must either be regenerated in real time, updated from a previous image or have been pre-rendered.

Regenerating the scene in real time is the approach chosen by most virtual reality researchers. This approach requires no special purpose virtual reality application dependant hardware. As a result Silicon Graphics work stations are often used as rendering engines. This technique is inherently wasteful as the active lifetime of an image decreases as interactive latency is reduced.

Updating a previous screen is a technique which is more efficient and faster than regeneration. Generally, updates are achieved by scrolling the image to compensate for user head rotations with the rendering engine filling the gaps created by scrolling.

Scrolling with 3 degrees of rotational freedom while compensating for wide angle viewing lenses requires substantial hardware within the display controller. Once built, this scrolling hardware allows no flexibility in the wide angle viewing lens mapping.

Pre-rendering an image may not at first appear to be a viable option, since motions of the user are not known in advance. However, it has been found that if a scene is rendered in a manner independent of the orientation of the user's head, it may be possible to decide the video memory fetch location of a pixel based upon the user's head orientation. As a result, pre-rendered images may be continuously reused without computational penalty. In order to create an image which is to be reused, the image created must be independent of the viewing orientation of the user's head. The task of orienting the image relative to the user's head is then dedicated to the display controller and is performed at pixel display time.

In order to generate an entire scene independent of orientation the image can be thought of as being rendered onto an encapsulating surface surrounding the user's head. A spherical encapsulating surface has been contemplated, in which the spherical surface is mapped into display memory using a spherical coordinate system, with uniformly spaced ϕ units mapping to the columns and uniformly spaced θ values mapped to the rows of the display memory. Such a mapping scheme for the sphere results in varied pixel densities within the display memory. As such, an ellipsoid or cube may prove to be a better choice of encapsulating surface when considering uniformity of pixel storage within display memory.

All user translations, lighting and shadowing changes, stereoscopy and object animation require some form of redrawing. Any form of redrawing falls into the category of scene animation. Although the rendering engine is responsible for scene animation, the display controller accommodates depth buffers. This allows different sections of the scene to be rendered to different display memories ordered by depth. For example scarcely changing background scenery may be rendered to a background buffer and updated very occasionally relative to the frame rate, maybe once every few seconds. This is possible because the background image is rendered independent of the user's head orientation. A foreground object which is undergoing vigorous animation may be drawn into a front buffer by itself. Since only one object needs to be redrawn as opposed to redrawing the entire scene, most of the rendering engine resources may be focussed upon drawing the one object. This results in a high update rate for the object and in very fluid animation.

Display generation generally consists of two processes. The first process interfaces the video memory to the CPU and is referred to as the rendering engine. This stage may be implemented in hardware or software. The rendering engine, in effect, draws the image into the video memory. The second process sends the image drawn by the rendering engine to the display device and is referred to as the display controller. Such a system is shown in Figure 3.

The purpose of a display controller within such a system is to provide a simple sequential counter which informs the video memory of the address of the next pixel to be displayed. Such a controller also determines the sizes and positions of the visible sections of the display as well as producing the horizontal and vertical synchronisation signals required by display devices.

Although many published projects focus on improving the performance of the rendering engine, or providing faster access to the frame buffer, little attention has been given to optimising the display controller for the needs of a virtual reality display system. In order to maintain and interact with high quality virtual environments, the described embodiments of the present invention focus on the display controller rather than the rendering engine. As a result a hardware system has been devised which is devoted to scene maintenance, allowing the rendering engines and available CPU power to concentrate on scene animation.

In effect, a display system constructed in accordance with embodiments of the present invention allow creation and storage of a scene in a virtual environment which comprises images encompassing any possible view from the user's position in the virtual world. In other words, the stored scene includes images of the portions virtual environment which are to the sides, behind, above and below as well as in front of the user. The particular image which is to be displayed to the user at any given time is retrieved from the scene storage corresponding to the direction in which the user's head is oriented. Since the entire scene is stored, changes in head orientation merely requires a different portion of the storage to be accessed. To provide for stereoptic viewing, separate scenes for the left and right eyes can be stored and accessed parallely. Also, depth buffering of different portions of the scene can be used, wherein relatively static background images are stored separately from relatively changeable foreground images, with the foreground image portions being overlayed on the background image portions at the time of display. Since the portion of the scene to be displayed to the user (the "viewport") need only be retrieved from storage, this operation can take place as memory address mapping procedure, synchronous with the pixel display rate of the display device. Therefore, the new display controller directly reduces latency to user head rotations by fixing user head rotation latency to the refresh period of the display device. Latency to other forms of interaction such as user translations through the virtual scene, head motion parallax and stereoscopy are indirectly reduced through the use of image overlaying.

Image overlaying or image composition is a technique often used by low–cost non head–mounted display graphics systems such as video games to increase the apparent realness of a scene. It is also often used by high performance systems to increase the maximum rendering rate. Different sections of the visible scene are drawn into separate display memories then overlayed to form a final scene (a similar technique is sometimes used when creating cartoons to reduce the drawing required per display frame). When translating through the virtual scene, each display memory may be updated independently. Close objects receive more of the rendering engine's time than less frequently changing background objects. Image overlaying generally does not work very well with conventional display controllers in virtual reality systems as user head rotations invalidate all display memories including those containing background images. This mass invalidation occurs because all of the images in the display memories are rendered with a fixed viewport mapping so when the viewport changes due to a user head rotation all of the images have an incorrect viewport mapping. The new display controller does not suffer the same drawback since the rendered scenes in display memory are independent of the viewport mapping orientation and do not become invalid when the viewport mapping changes. The only exception to this is when providing stereoscopy for large orientation changes. Some re-rendering of close objects may be required for an accurate stereoscopic view of the scene.

Figure 4 shows a block diagram of a known image generation and display system 10 which performs two processes represented by cycles 12 and 14. The first process, called rendering, draws images into a display buffer memory 16, and is performed in the update cycle 12. The second process sends the image data in the display buffer memory 16 to a display device 18, and is performed by a display controller operating the refresh cycle 14. As can be seen from the block diagram of Figure 4, the rendering process in the update cycle can be considered as comprising a number of individual steps 20 to 24, including accessing a shape database (20), geometric transformation of the shapes retrieved (21), acceptance or rejection of transformed shapes (22), determination of shading and reflection characteristics depending upon perceived light (23), and clipping of the transformed shapes forming the image (24). After rendering of the image the viewport mapping takes place at stage 26 the final construction of the image to be displayed is formed, and is then scanned at stage 30 for conversion to a format which is applicable for display on a raster scan display device 18. On the other hand, the refresh cycle 14 is carried out synchronously with the image display rate on the display device 18, which merely comprises addressing the pixels in the display buffer memory in accordance with the raster scanning sequence (stage 32), and comprising the image from the various image layers of stage 34 (described in greater detail hereinafter) to pass each pixel value to the display device 18.

From the above, it can be ascertained that an update cycle has occurred when all required primitives in a scene have been drawn into display memory, whereas a refresh cycle has occurred when an image in the display memory 16 has been sent to the display device 18. Conventional virtual reality display systems bind the latency associated with user head rotations to the period of the update cycle. While this has the obvious advantage of not requiring any specialised hardware, latency tends to suffer as the update cycle period can often be lengthy, unpredictable and dependent on scene complexity.

The delayed viewport mapping display system architecture of a preferred embodiment of the present invention makes it possible to bind rotational latency to the refresh cycle period which tends to be short, fixed in length and independent of scene complexity. Figure 5 is a block diagram of such a display system, showing how the viewport mapping stage 26 has been shifted from the update cycle 12 of display system 10 to the refresh cycle 14' of the new display system 10'. The rendering process of the update cycle 12' in system 10' is essentially equivalent to the rendering process of system 10, however no mapping of the image onto the viewport takes place during the update cycle 12'. Instead, the refresh cycle 14' has been altered to accommodate mapping of the rendered image onto the selected viewport, with reference to the user head orientation data available, during the image refresh cycle. The refresh cycle 14' shown in the display system 10' in Figure 5 also has some additional stages which are described in greater detail hereinafter. Namely, the pixel addressing stage 32, which relates to the scanning screen location for the display device 18, provides output to a wide angle lens correcting stage 38. The wide angle correction stage 38 is described in detail below and may comprise, for example, a look-up table with an entry for each pixel of the display device 18 to provide compensation for wide angle lens components which may typically be provided in virtual reality viewing equipment. Following generation of a pixel address (stage 32), and the wide angle correction stage 38 (if required), is the viewport mapping stage 26 which received head orientation input data from the head tracking apparatus. The viewport mapping stage in this instance generates a vector which can be utilised by the pixel location stage 40 to identify the address of a pixel to be displayed in the display buffer memory 16. Thereafter, the image composition takes place by overlaying various image layers (stage 34), as described in greater detail below, before passing an output signal to an anti-aliasing stage 36. The anti-aliasing process carried out at stage 36 immediately prior to display on the display device at stage 18 ensures that noticeable aliasing (i.e. the occurrence of jagged lines and edge effects), which may result from sampling quantisation, is produced in the final displayed image.

In the display system 10′, since viewport mapping is performed after rendering, the image in display memory should completely encapsulate the user's head. As mentioned previously, many different encapsulating surface models may be used, the conceptually simplest perhaps being a spherical surface centred at the user's viewpoint. In this case, therefore, the latency associated with user head orientation changes is determined by how long it takes to select the required portion of the encapsulating surface and write the image data to the display device. Where the encapsulating surface is chosen to be spherical, the selection of the appropriate region of the sphere to display can be modelled as projecting rays from the viewpoint onto the surface of the sphere so defining the area to be seen. Alternatively one can view the process as one of changing coordinate systems from a real world to an encapsulating (e.g. spherical) world. The coordinate transformation indeed requires a great deal of computation for each pixel to be displayed, and would most likely be infeasible using present day software techniques. However, the approach is ideal for a pipelined hardware implementation which, while using relatively fast hardware components, has a fairly simple structure and can be built relatively economically.

Investigations of a hardware design based on a spherical system have been found to have some annoying characteristics. Firstly, in mapping a sphere onto a memory array one tends to waste a lot of memory or else have a very complex mapping scheme. Second, the apparent sizes of pixels in a sphere vary greatly as one moves from equator to the poles. Third, the coordinate transformations involve trigonometric functions which are moderately expensive to implement. By far the dominant issue is the first in that much more display memory is required in a spherical topology than a conventional graphics system, and since accessing in a regular pattern is not possible, it is necessary to use relatively fast and expensive memory.

The basic delayed viewport mapping model 10′ (Figure 5) will essentially work for any surface surrounding the user. A spherical surface is intuitively simple in that every point is equidistant from the user at the centre, however other surface topologies, such as a tetrahedron have also been contemplated. Eventually, a cubic encapsulating surface model topology was chosen. A cubic surface has advantageous properties in that no memory is wasted since the constituent surfaces can be mapped directly onto a two dimensional memory array as found in conventional memory chips. The coordinate transformations are linear and can be implemented essentially with a matrix multiplication, with intersections with a cube being easier to determine than for a sphere. Pixel sizes do not vary as much as for a sphere. While there may be some concern about strange effects occurring in corners and edges these have been shown not to be significant.

In view of the above, the surface of a cube centred at the user's head is the chosen topology for storing images within the display memory, which is described in detail hereinafter. Figure 6 shows a flattened scene constructed from a cubic surface topology as it is stored in display memory (left hand image) and as an image from the scene at an arbitrary orientation (right hand image). Note that the distortion of the lower right hand side of the pool table in the display memory scene is not seen in the image created for display.

Traditionally the display controller (see Figure 3) performs relatively simple computations, however the operations performed by the controller may be made more complex if the computations can be pipelined. For currently available medium resolution graphic displays devices with resolutions in the order of 640 pixels horizontally and 480 pixels vertically, a new pixel must be displayed approximately every 40 nanoseconds.

The display control and display output processes may in fact be considered as a two stage pipeline in which the first stage generates the address of the current display pixel for the video memory and commences the memory access sequence while the second stage sends the output of the previous access to the physical display device. Traditionally this pipeline is two stages long however there is no reason why the display pipeline may not be extended beyond two stages so as to include complex operations such as non-sequential pixel address computations, depth buffer comparisons and the like, so long as the pipeline throughput is comparable with a desired display rate for the pixels. Within the presently described implementation the pipeline, referred to as the address recalculation pipeline, is extended to eight stages. Each stage or group of stages is discussed hereinbelow.

The address recalculation pipeline 100 forms the pixel addressing stage 32, the wide angle correction stage 38, the viewport mapping stage 26 and the locate pixel stage 40 of the refresh cycle 14' of the display system 10'. The pipeline 100 is described hereinafter to include display buffer 16, the image composition stage 34 and the anti–aliasing stage 36 in order to provide RGB pixels to the display image device 18.

In view of the pixel display rate mentioned above, the address recalculation pipeline must have a throughput of 1 result every 40 ns once the pipe is full. As there are no stage interdependencies within the pipeline no pipeline stalls occur so pipeline filling need only occur once. In our implementation we use the horizontal retrace time as well as the time the raster spends in non–display regions of the screen to fill the pipeline. The pipeline takes approximately 300 ns to fill which is a delay one order of magnitude less than the horizontal retrace signal. This means that there is plenty of time to fill the pipeline.

The address recalculation pipeline 100, as show in Figure 8, uses hardware to perform orientation viewport mapping after a scene has been rendered.

A significant advantage of the pipeline 100 is that the update rate for user head rotations is bound to the update rate of the display device (usually 60+ Hz) instead of the rendering frame rate. Also, the latency associated with the address recalculation pipeline does not include rendering time and double buffer swap delays. The orientation of the view the user sees does not need to be known until the first pixel is to be sent to the display device. This means the images presented to the user is based on the most up to date head tracking information. The nature of the latency related to head rotations is depicted in Figure 7.

In order to perform viewport mapping after rendering, the rendered view must completely encapsulate the user's head, so when the user's head orientation changes the view from the new orientation has already been rendered and the address recalculation pipeline presents the user with the new view computed from the pre-rendered image in the system's display memory. As mentioned previously, the surface of a cube was chosen as the encapsulating rendering surface which results in a moderately simple hardware implementation and has fewer aliasing artefacts than most other surfaces. The cube's greatest advantage however is in its associated rendering simplicity. The surface of the cube may be rendered to by rendering to six standard viewport mappings (up, down, left, right, front and back). Thus most standard rendering algorithms require little or no modification. As mentioned, Figure 6 depicts a virtual environment scene rendered onto the surface of a cube together with an image from a particular orientation created by the address recalculation pipeline 100 from the rendered scene.

The architecture of the address recalculation pipeline 100 differs from mainstream architectures in the nature of the pixel addressing mechanism. In a conventional display system, pixels to be displayed on the output device are fetched from the display memory sequentially. All logically adjacent pixels in the display memory appear adjacent on the display device. The address recalculation pipeline is different in that rather than fetching pixels sequentially, pixels are fetched from display memory based on the pixel's screen location and the orientation of the user's head. The distortion due to wide-angle viewing lenses can also be compensated for.

The address recalculation pipeline 100 runs at video display rates with the main high-speed input 110, as shown in Figure 8, being the X-Y location of the pixel on the display device. The output 132 from the pipeline is a stream of RGB pixels. The first stage 112 of the pipeline converts the screen location of a pixel from the input 110 into a three dimensional vector 114 pointing in the direction at which the pixel is seen relative to the user's head as seen through the wide angle viewing lenses. The next stage 116 of the pipeline multiplies this vector by a matrix containing user head orientation information received from a head orientation input 118. The output of the matrix multiplication stage 116 is another three dimensional vector 120. This new vector points in the direction at which the pixel is seen relative to the world coordinate system. The third stage 122 of the pipeline converts the three dimensional vector 120 into a display memory location 124. Next, four adjacent pixels are fetched from the display memories 126. Finally the four pixel sets are composed and blended at stage 130, using redundant bits from the matrix multiplication stage. The resulting anti-aliased pixel 132 is then sent to one of the output devices 134 in the head-mounted displays. The hardware is duplicated to obtain a stereo view.

The X-Y screen location input 110 of the current pixel to be displayed is provided to the pipeline 100 by a conventional graphics controller at a normal pixel display rate. A look-up table called a Wide Angle Viewing Lens Look Up Table is provided at stage 112 to convert the pixel's screen X-Y location into a three dimensional unit cartesian vector 114 pointing in the direction at which the pixel is seen by the user, through the wide angle lenses, relative to the user's head. The look-up table has one entry per display pixel where each entry consists of three 16-bit vector components. For a display device of resolution 640 by 480 pixels the look-up table will require a memory of 1024(cols)*512(rows)*6(bytes per entry) = 3 Mbytes. Many head-mounted displays use wide angle viewing lenses which preserve a standard viewport mapping, however if a special mapping is required for higher fields of view, the look-up table may be loaded with a new lens mapping, compensating for the lens distortion without run-time penalty. Figure 13 shows views created from the display memory image in Figure 6 with different wide angle viewing lens mappings.

The 48-bit output of the wide angle viewing lens compensation stage 112 feeds into a matrix multiplier which forms the next stage 116 of the pipeline. The multiplier multiplies the pixel direction vector with a 3 × 3 matrix containing user head orientation information. The resulting output vector 120 points in the direction at which the pixel is seen by the user, through the wide angle viewing lenses, relative to the world coordinate system. The pixel direction vector 114 is fed into the matrix multiplier 116 at pixel display rates while the head orientation matrix is updated from the head orientation input 118 at the start of each display frame (i.e. after each vertical sync signal). The matrix multiplier is also implemented with 16-bit fixed point arithmetic and is built with nine commercially available 16-bit by 16-bit, 40 ns multipliers and six 16-bit, 40 ns adders. The output vector 120 from the matrix multiplier 116 is in the form of three 16-bit fixed point vector components [Xo Yo Zo].

Figure 11 illustrates the matrix multiplication stage 116 of the pipeline 100. The matrix equation to be solved by the matrix multiplier stage is indicated by reference number 150 in Figure 11, wherein Xi, Yi and Zi represent the vector 114 of the current pixel location relative to the axis of the head-mounted display device. The quantities Xx, Xy, Xz, etc. in the 3 × 3 matrix of equation 150 are quantities input to the matrix multiplier 116 from the head orientation input 118, and represent vectors aligned with the orthogonal axes of the head-mounted display device relative to the axes of the encapsulating surface topology. The hardware implementation for this stage required three multipliers and two adders for each orthogonal axis, and a block diagram of a suitable construction is shown at 152 in Figure 11. The result of the matrix multiplier 116 is an output vector of the quantities Xo, Yo, Zo which represent a vector in the direction of the current pixel relative to the encapsulating surface axes.

In order to understand how the vector conversion stage 122 of the pipeline operates the topology of the display memory must be known. A requirement of the display memory is that it should completely encapsulate the user's head. The topology chosen to encapsulate the user's head within this architecture is the surface of a cuboid with side length of two units, centred at the coordinate axes origin. This means the display memory is divided into six faces where each face contains a two dimensional grid of pixels. When folded correctly these faces form a complete surface around the user's head.

A run-time advantage of selecting the surface of a cube over the surface of a sphere is that the scan conversion process for the cube requires minor modifications to existing scan conversion/shading algorithms while scan conversion for the spherical surface requires major, computationally expensive changes to common algorithms.

The conversion process for a cuboid topology involves computing the point at which a ray intersects with the surface of a cube. When the cube is aligned to the axes of the coordinate system such that each face of the cube has one of its X, Y or Z coordinates fixed at +/−1.0, the intersection may be computed with a set of parallel divisions with range checks on the outputs of the divisions. For example, if the result of the divisions Xo/Yo and Zo/Yo are both within the range (−1.0, 1.0) the ray must intersect with only two of the six faces. The sign of Yo is then used to determine the face of intersection. The point of intersection of the face is then (Xo/Yo, Zo/Yo). The divisions must occur at pixel display rates, so the divisions are performed by a reciprocal look-up followed by a normal multiply using another set of 40 ns multipliers. The reciprocal look-up has extra output bits which are used to compensate for classification with fixed precision arithmetic. A programmable logic device is used to accumulate data from the appropriate data paths to multiplex the divider outputs to form the display memory address.

Figure 12 is a block diagram of one possible form of vector converter 140. Finding dividers which operate at video speeds is quite difficult and the process is further complicated by the fixed point nature of the pipeline. Consequently a table look-up followed by a multiplication was the division method chosen for this architecture. Sixteen bit precision inputs/outputs require 128 Kbyte look-up tables 142. Using a look-up table means that the range of outputs may be bounded between ±[1.0 and 2.0], simplifying preliminary face classification (a candidate face must have a divisor in the range ±[1.0 to $\sqrt{3}$]). Adders 144 are used post division to simplify signs for range checking. A programmable logic array 146 is used to classify face intersection from range information from all of the appropriate data paths. Aligned fixed point arithmetic with bounded reciprocal look-up tables means that all required range information is stored within the top bit of each data path. The programmable logic array 146 and selector 147 are structured so as to perform the following algorithmic function:

```
if (! ((b15_xdivz^b13_xdivz) ¦¦ (b15_ydivz^b13_ydivz) ¦¦ out_of_rangez))) {
        get_surrounding_pixels (xdivz, ydivz, (b15_z) ?0:1);
}
else if (!((b15_xdivy^b13_xidvy) ¦¦ (b15_zdivy^b13_zdivy) ¦¦ out_of_rangey)) {
        get_surrounding_pixels (xdivy, zdivy, (b15_z) ?3:2);
}
else if (!((b15_zdivx^b13_xdivx) ¦¦ (b15_ydivx^b13_ydivx) ¦¦ out_of_rangex)) {
        get_surrounding_ pixels (zdivx, ydivx, (b15_z<0) ?5:4);
}
```

wherein the function get_surrounding_pixels has the structure
get_surrounding_pixels (int x, int y, int face)

where	int x is the x coordinate of the intersection
	int y is the y coordinate of the intersection
and	int face is the face of the cuboid surface which contains the intersection point,

and wherein

b15_z	=	bit 15 of z
b15_xdivy	=	bit 15 of x divided by y
b13_sdivy	=	bit 13 of x divided by y
out_of_rangez	=	is out if range to form a reciprocal (i.e. $\lvert z \rvert < 0.5$)

Essentially, the PLA 146 performs the logic of the if statements and the selector 147 performs the operand selection for the operands of the get_surrounding_pixels function.

The current hardware implementation of the address recalculation pipeline 100 uses 16-bit fixed point arithmetic for all computations. The system is designed for medium resolution displays of the order of 640 by 480 pixels with a 60 Hz refresh rate which means each pipeline stage must complete in 40 ns. As mentioned above, the wide angle viewing lens look-up table requires one 48-bit entry per display pixel, and the resulting look-up table is 3 Mbytes in size. The system may accommodate many different wide angle lens types by downloading different distortion mappings into the look-up table. The matrix multiplier stage is implemented with nine 16-bit by 16-bit commercially available multipliers and six 16-bit adders. Some additional 16-bit registers are used for pipeline synchronisation. The vector conversion stage which converts a three dimensional vector into display memory location requires six 16-bit divisions, some programmable logic and buffering circuitry. The divisions are implemented with a reciprocal table look-up followed by a multiplication.

The circuit diagram for the pixel addressing, wide angle correction, viewport mapping and locate pixel stages 32, 38, 26, and 40 of the address recalculation pipeline 100 is illustrated in Figures 37 to 49, the input being lines PA0-17 in Figure 37 and the output being lines FX0-15 and FY0-15 in Figure 48. A legend which specifies each of the chips used in the circuit is provided on page 73.

The vector conversion stage produces a display memory location. The display memory 126 itself is organised into six faces, where the faces logically fold to form a cube. Each face of the cube has a display resolution of 512 by 512 pixels. This display resolution results in 1.5 Mpixel display memories. Each display memory is z-buffered and double buffered with a separate rendering engine. A stereo system employing multiple display memories for image composition requires vast amounts of memory. To implement high resolution display memories with current technology, static memory is used due to the speed requirements and the non-sequential nature of the memory access. As a result, the cost of display memory tends to dominate the cost of the system. This may change as new memory chip architectures become available.

The pipeline orientates the address generation process within display controller with the orientation of the user's head as the pixels are being fetched from video memory. This is the latest stage at which the user's head orientation may be introduced into the view computation within a virtual reality head-mounted display system. As user head orientation is not introduced until the pixel is being drawn, the effective computation latency for user head rotations is theoretically reduced to the time it takes one pixel to pass through the address recalculation pipeline. In our simulations however, we have considered the user's head to be stationary for an entire display frame. As a result the average latency for head rotations, L_r, for each pixel is given in the equation below, assuming the head remains stationary for the duration of the frame.

$$L_r = (F + T)/2 + (C + P)$$

where L_r = Average pixel display latency for user head rotations.

F = Time to draw one frame.

T = Time between updates from tracking device.

C = Cost of computing head position.

P = Latency of Address Recalculation Pipeline.

Using a standard raster based display device and a high-speed magnetic head tracking device the average rotationally based latency L_r has been found to be in the region of 12 milliseconds and the maximum latency approximately 24 milliseconds.

Figure 9 is a block diagram of a portion of an address recalculation pipeline 100 for implementing delayed viewport mapping of an image mapped onto a spherical encapsulating surface, and many of the features described hereinbelow in connection with the spherical mapping pipeline are equally applicable to a cuboid surface topology. A first input to the pipeline intercepts the output of the next_pixel_pointer register containing the screen location of the next pixel to be displayed. The first stage of the pipeline converts this location to a 3D vector pointing in the direction of the pixel as seen by the user within the head-mounted display. Typically this stage is a large, high-speed look-up table. As the output of this stage is a vector in the direction of the pixel as seen by the user, this vector may be altered to correct for viewing perturbations caused by wide angle viewing lenses. For this reason the first stage of the address recalculation pipeline is referred to as the Wide Angle Viewing Lens Look Up Table (WAVLUT).

Stages two and three of the pipeline form a matrix multiplier. One input matrix consists of three orthogonal unit vectors describing the orientation of the user's head relative to world coordinates. This matrix is the second external pipeline input, and is usually updated with the user head position at the start of a display frame. The other input to the matrix multiplier is the current pixel's direction vector output from the previous pipeline stage. The inputs to the multiplier are the pixel direction relative to the user's head and the orientation of the user's head relative to the real world. As such the output of the multiplier is a 3D unit vector describing the pixels direction relative to the world coordinate system.

Stage four and five consist of a set of Cartesian to Spherical coordinate converters as commonly used within digital signal processing applications. Commercially available converters operate at 25 Mhz.

Figure 10 provides a block diagram overview of the remaining portion of the spherical mapping address recalculation pipeline. The display memory is accessed in stage six of the pipeline where multiple display depths are accessed concurrently. Each display memory depth passes a block of four adjacent RGB pixels to the tree depth comparison stage and anti-aliasing stage.

It is desirable for the address recalculation pipeline to take into account the shape of the lens used for wide angle viewing. Images may cover as much as 140° horizontally and 100° vertically, often without covering the field of view linearly. It may be possible to find a lens mapping that could be encoded in hardware that could compute corrections in real time, however, such hardware systems would be bound to the lenses used and the mapping it provides. Constraining the display hardware to a particular lens is undesirable in terms of flexibility, as most manufacturers of virtual reality head-mounted displays tend to use completely different lens systems, providing vastly varying display pixel densities for different eye view positions.

The solution for providing wide angle lens compensation used by forms of the present invention requires a look-up table within the address recalculation pipeline. When the hardware is initialised the optical lens mapping is downloaded into this look-up table. Compensation for visual perturbations caused by wide angle viewing lenses may be expensive to compute when all the corrections are handled by the rendering engine, so storing the pre-computed correction for each pixel within a look-up table means the rendering engine becomes largely independent of the wide angle viewing lenses. Removing this overhead from the rendering engine is an extremely desirable feature.

The wide angle viewing lens look-up table has an entry for each pixel in the display. For a resolution of 640 by 480 pixels, the look-up table requires a 19-bit address. Each entry consists of two vectors, where a vector has three components of 16-bit accuracy, resulting in each entry being 96-bits or 12 bytes long. As a result the WAVLUT is built from 6 megabytes of 35 ns static memory.

The matrix multiplier stages consist of two parallel matrix multipliers used to orientate the user's head position to the world orientation of the display memory. The precise computation is displayed in Figure 11, as discussed briefly above. The left hand side 150 of the figure depicts the form of the multiplication while the right hand side shows the hardware implementation 152. The matrix multiplier consists of nine multiplication units and six addition units. The multipliers with operating delays less than the required 40 ns may be purchased reasonable prices. The output of each multiplication unit it latched internally before going into the adder units which form the second stage of the multiplication pipeline.

The unit vector $[Xo\ Yo\ Zo]^T$ on the right hand side the of matrix multiplication 150 is the output of the WAVLUT. This vector points in the direction at which the pixel is seen by the user relative to the head-mounted display. The 3×3 matrix on the left hand side of the multiplication 150 contains three orthogonal unit vectors orientated in the direction of the user's head relative to the world coordinate system and is supplied directly from the user head tracking equipment. Once the orientation matrix is multiplied by the pixel direction vector $[Xi\ Yi\ Zi]^T$ a resulting output vector is produced.

The Cartesian to Spherical coordinate converters 117 (Figure 9) consist of a set of commercially available digital signal processing devices. These devices accept a cartesian $[X\ Y\ Z]^T$ vector where each component has 16-bit accuracy and produces a spherical $[R\ \theta\ \phi]^T$ vector where θ and ϕ are each of 12-bit accuracy. These components are then used as row and column addresses for the display memory.

For a display memory which emulates a spherical encapsulation surface, the surface may be organised on a latitudinal and longitudinal basis, and depth buffering can be alternatively be implemented by effectively placing one sphere inside another sphere and defining a pixel intensity for which the pixel is to be transparent. Black can be chosen to represent a transparent pixel. From this definition, when a pixel on a sphere's surface is black in colour, a pixel which appears directly behind the current pixel is to be displayed. All the pixels behind the current pixel will have the same direction as the current pixel, hence the same memory location. The only difference is that the pixels behind the current pixel will appear within the display memory surfaces representing greater depth. Stage 6 of the pipeline portion illustrated in Figure 10 contains a plurality of display memories 127 for each of the left and right views. Each display memory is for storage on an image representing a particular layer in the overall image to enable the depth buffering technique to be implemented. Each of the memories in the left and right sides are addressed using the same address coordinates and the accessed pixel values are passed to the depth comparison and anti-aliasing stage 129, where the pixels from the various layers are compared to determine which one is to be passed to the display device 131.

The resolution chosen for a display depth generally involves a trade-off between display memory cost and system performance. The resolution must be sufficient to allow oversampling even within the display regions which have increased pixel densities due to the wide angle viewing lenses. The degree of oversampling is an important factor. As the oversampling ratio is increased the anti-aliasing performance is increased however the display memory cost increases and rendering time increases.

The number of pixels within a display depth is $2n^2$, where n is the number of latitude lines on the surface of the encapsulating sphere. To implement double buffering and z-buffering rendering techniques, each pixel requires two colour fields and one Z field. With 16-bit colour and 16-bit depths, each pixel requires six bytes of memory. This means the memory requirement per depth per eye is $12n^2$ bytes. If an image contains one thousand and twenty four latitudinal lines, a total of twelve megabytes of 35 ns ram per depth per eye is required. One thousand and twenty four latitudinal lines seems to provide an adequate trade-off between display memory cost and anti-aliasing performance for a 640 by 480 display system.

The display technique described hereinabove is effectively a double sampling technique where the first sampling quantisation occurs when the encapsulating image is generated and the other sampling quantisation occurs when the image to be displayed on the output device is created at pixel display time. A result of this double sampling is noticeable aliasing, that is the occurrence of jagged lines and edge effects (edges of polygons breaking up). Although not critical to system performance, the occurrence of these effects is annoying and as a result some hardware has been designed to reduce the effects of aliasing and edge effects.

A commonly used technique to anti-alias images, especially within ray tracing algorithms, is to drastically oversample the image (say by a factor of 16), then reduce the image to the sampling rate of the display device by applying some form of filter. Extreme oversampling of a pre-rendered scene drastically increased the requirements placed on the display memory and the rendering engine. As a result extreme oversampling may not be a practical solution to the aliasing problem.

The four pixels of each display depth are passed into a tree comparator 129 which compares each of the four pixels independently against the corresponding pixels from other depths to determine which pixels are transparent. Alternatively, a pipelined comparator can be used instead of tree comparator 129, and is in fact preferable for a large scale system. The front-most non-transparent pixels are then piped into the anti-aliasing filter then onto the display device. This technique forms the basis for the anti-aliasing method and apparatus described below.

With respect to the performance of the pipeline architecture, latency to head rotations (except stereoscopy) is fixed to the display rate. This means the eye may track an object as the head rotates with little detectable latency. A display update rate of 60 Hz combined with a head tracking update rate of 120 Hz results in an average rotational latency of approximately 20 ms. Forward prediction of head location using a linear or higher order models may reduce this latency even further. As the viewport mapping process does not have to pass through the double buffer display switching (as it is independent of rendering), the time at which the raster passes through the centre of an image is known precisely. This in turn improves the quality of the head predication process.

Providing a binocular stereoscopic view of the scene may require some objects to be re-rendered. Head rotations cause stereoscopy to diminish at ±90° of the original orientation, however due to the nominal intraocular distance of 6.3 cm for humans, only relatively close objects translate significantly, so image composition allows these objects to be re-rendered independently. Added to the fact that the human visual system takes several tens of milliseconds to perceive a stereo scene, and apparently is not sensitive to small temporal errors the latency involved in re-rendering for stereoscopy is important but not critical to system performance.

Latency to translations is often less noticeable than latency to rotation as many translations are the result of a user gesture such as pointing a finger or pushing an accelerator rather than actual physical motion. Providing an adequate update rate for translation however is important to provide a sense of fluid motion. An update model can be used so different objects may be re-rendered to different display memories in order to reduce the overall load on the rendering engine. The update model can be based on the size, distance and relative velocity of a particular object. Clustering algorithms may also be used to tie objects with a similar update rate to the one display memory.

The preferred embodiment uses static RAMs exclusively for all of its memory requirements, however, the two most common forms of memory used within a conventional graphics system are DRAM and VRAM.

DRAMs are generally the cheapest memory to buy, but they are often very slow. These memories consist of a large array of single transistor memory cells. While DRAMs were used in the earliest graphics system, their lack of speed requires quite a large degree of interleaving when used in modern systems. VRAMs overcome the lack of speed suffered by DRAMS in sequential accesses by providing an additional shift-register inside the chip, which reads one row of the memory at a time then shifts the data out sequentially via a separate external port.

Within an address recalculation pipeline (ARP) system 100 random accesses occur at pixel display rates. Due to this random nature the techniques for interleaving DRAM are not effective. DRAMs typically have access times in the order of 60 ns when operating in page mode and over 120 ns in random access mode. So even in page mode the DRAMs are not fast enough to keep up with the ARP. It is possible to use redundant DRAMs to increase the apparent speed, however after introducing a factor of 2 redundancy (i.e. each display memory has two identical copies that are accessed independently), the cost effectiveness of the DRAMs becomes similar to the totally static solution. VRAMs are only capable of sequential access through the high speed port so they are no more effective than DRAMs.

While VRAMs and DRAMs are no more cost effective than static RAMs, there is a new type of dynamic RAM available. While this form of memory goes under several different names, EDRAM, RAMbus, cachedRAM etc, the principal of operation behind them all is similar. Basically these memories are similar to conventional DRAMs except the sense amplifiers at the bottom of each column have a static cache. When a new row in the memory is accessed, the entire row is copied into the high-speed on-chip static cache. Upon further accesses to the same row, the data is written or read from the high-speed cache rather than the slow sense amplifiers. Typically once a row has been accessed subsequent accesses to the same row only require 15 ns. While these new memories have different names from different manufacturers, the following discussion will be based on the use of EDRAMs (Enhanced Dynamic RAMs) which are available from RAMTRON Inc., as described in the DM2223/2233 Sync Bursting EDRAM 512 Kb × 8 Enhanced Dynamic RAM Data Sheet, Ramtrom International Corporation 1994, pp. 24. These memories only require five percent more silicon area than conventional DRAMs and are approximately twenty percent more expensive than conventional DRAMs of the same density.

Conventional external caching system used with DRAMs are unsuitable for this application as most display memory accesses would result in a cache misses. Conventional caches are most effective when accesses to the same memory location occur frequently. Memory access made by an ARP tend to be local, but they are almost never the same as the most recently accessed memory locations. The cache fill characteristics of an EDRAM are completely different to conventional external caches, and it is the high cache fill bandwidth of the EDRAM that makes them useable.

The following considers how EDRAMs may be used to replace the SRAMs in the preferred embodiment.

The path the memory accesses from an ARP into the system's display memory follow is non-sequential and often complex. Factors such as the user's head orientation, wide angle viewing lens distortions and display memory topology all affect the nature of the path the accesses follow. Without assumptions about the user's orientation or the lenses, the display memories are unable to predict the location of the next memory location (it requires an ARP to do that). While the display memory accesses are unpredictable, there tends to be some locality in the accesses. That is the Euclidean distance between accesses tends to be quite small. Therefore if cached lines of the memory can be arranged into blocks where each block covers a large area, it is quite likely that successive accesses will occur in the same block due to the small Euclidean distance.

First we must arrange the caches so that the probability of a cache miss is minimised. As the display memories make no assumptions about the direction of accesses, a square cache topology is used. This means the rows of the EDRAM must represent a square block. This may be achieved by exchanging the high order bits of the row address with the low order bits of the column address of the memory access. Thus if only the low order bits of the row and/or column address from the ARP change for a subsequent access, the pixel referred to in the subsequent access is currently held in the cache of the EDRAM and may be accessed in 15 ns. As a single face is spread across four display memories (when an interpolation filter is used), each of the lines in the EDRAM 280 is 32-pixels by 32-pixels, as shown in Figure 34. Such an arrangement may also be advantageous when rendering and inherently solves many of the problems discussed in Deering M., Schlapp S. and Lavelle M., 1994, "FBRAM: A new form of Memory Optimized for 3D Graphics", *Proceedings of SIGGRAPH '94 (Orlando, FL., 24-29 July, 1994), In Computer Graphics, Annual Conference Series 1994*, pp. 167-174. That is, this cache is suited to the memory access locality observed when scan converting polygons.

The ARP graphics system 100 requires a pixel every clock cycle. This is not a problem until an EDRAM cache miss occurs, a strategy is needed to deal with an EDRAM cache miss. Note that a cache miss occurs when any one or more of the simultaneous 4-adjacent-pixel fetches suffers a cache miss. When an access to the display memory misses the cache, a full cycle read of the DRAM in the EDRAM must occur. The penalty for a DRAM read depends on the nature of the previous access. If the previous access was a cache hit, the penalty for a cache miss is 35 ns, if the previous access was a cache miss, the cache miss penalty is 65 ns. This is because the dynamic RAM row pre-charge can occur during a cache hit, so if a cache miss follows a cache hit the row pre-charge of the cache miss is hidden by the previous cache hit.

This represents something of a problem for the ARP as a 65 ns access may stall the pipeline. A simple solution is to place a First In First Out (FIFO) buffer 282 between the first part of the ARP 100' and the display memory 16, as shown in Figure 35. Thus if some memory accesses causes cache misses, the ARP 100' may buffer its output in the FIFO 282 unaware of the cache miss. While placing a FIFO 282 between the ARP 100' and the display memory 16 hides cache misses from the ARP, the cache miss is not hidden from the output device. As such, another FIFO 284 is placed on the output of the EDRAM 280 used to form the display memory 16. This allows the ARP and the display device to maintain one result per clock while a cache miss occurs. If an entire line is buffered between the two FIFOs on either side of the EDRAM, the memory may be accessed during horizontal retrace periods.

It is possible to buffer an entire line between the two FIFOs on either side of the EDRAM. This allows the memory to be accessed during horizontal retrace periods and the only penalty is the addition of one display line to the interactive latency. Typically this additional latency is less than 32 microseconds and may be ignored. The main advantage of accessing the memory during horizontal retrace signals means that it is possible to use memories which have a slower access cycle than the ARP's clock cycle. This case may occur when complex lenses are used within the head-mounted display (HMD) causing a large number of cache misses, or the speed of the pipeline is increased. In the case of a standard VGA signal the ARP and the SRAM are only active for 80.6% of a complete display refresh cycle and are idle for 19.4%. The EDRAMs may use this time to catch up when cache misses occur. Figure 36 shows the periods when the ARP and SRAM are active during the first scan line compared with the amount of time the EDRAM may be active.

Simulations have revealed that when an entire line is buffered by FIFOs many common lens distortions may be used. These simulations involved computing the average and the maximum number of cache misses during the access to a line while it is being updated for 100 arbitrary orientations with different distortions. The table below shows the number of cache misses encountered with various wide angle lens configurations tested at one hundred arbitrary orientations. In the worst case, 8.14% of the accesses for a particular line resulted in a cache miss. In the SVGA example, the amount of time allocated to handle these cache misses is (8.14% * 80.6) + 19.4 = 26% of a scan line. As such, for the given cache miss rates, heavy cache miss penalties may be tolerated.

CASH MISS RATES

view distortion	**cache misses average**	**cache misses worst line**
fish-eye distortion 140 degree field of view	4.43%	8.14%
100 degree field of view	2.48%	5.47%
70 degree field of view	1.7%	3.75%

Catastrophic cache misses are a rare case in which the path of a scan line follows a cache block boundary and rounding errors in the ARP cause it to jump between cache blocks each memory access. Such a situation would result in a full cycle access each memory read. In simulations, catastrophic cache misses caused 200 cache misses for a single line out of a possible 640. EDRAMs available from RAMTRON would not suffer from catastrophic cache misses as they actually cache four lines internally rather than a single line. Although the blocks cannot be filled simultaneously, the scenario described previously would only suffer from 2 cache misses for 2 adjacent blocks rather than a cache miss each access. The results in the above table assumed that the EDRAM had a four line cache strategy.

A technique which has been shown to work well in accelerating computer graphics systems is image composition 34. Instead of treating the scene as a single entity it can be broken down into component objects and combined at the end to produce the scene. These component objects can be rendered separately and, if multiple renderers are available, the rendering process for the components can be performed in parallel. The big gain is that objects which do not change can be left and not rendered again even if other parts of the scene change. Thus a greater than linear speed-up effect can be achieved. Unfortunately this technique has been of little use in virtual reality applications since, as discussed above, a simple rotation causes every object in the image seen by the user to change.

On the other hand, the benefits of image composition can be enjoyed by employing the viewport independent approach for virtual reality applications introduced herein. Since the notional display surface encapsulating the user does not change when the user rotates, the objects already rendered can remain. The z-values of pixels are used to choose the closest pixel to the user as the one to be displayed. Hence rendered objects are not forced onto particular cubes in relation to their apparent depth, but instead the hardware is allowed to resolve dynamically which pixels to display. Separate rendering engines are provided for each cube. Thus it makes sense to distribute objects over cubes in such a way that those that need to be updated at similar times are grouped together. In this way the rendering capacity is not wasted on redundantly re-rendering objects which have not changed. A technique herein referred to as prioritised rendering which forms part of the database traversal or access stage 20 has been developed to handle display updates efficiently.

The technique of image composition 34 is depicted diagrammatically in Figure 14. Image composition occurs as pixels are fetched from the display memories to be displayed on the output device. All of the pixels from the display memories which correspond to the same screen location are compared to find the closest pixel, which is then displayed. Each stored pixel value has a plurality of colour components (such as a red, a green and a blue component) and may also be stored with an additional component which indicates the "distance" to the pixel within the context of the scene in which it is included. The distance component of the stored pixel valve is referred to as a z–value, and is sometimes used in conventional graphic display systems to perform hidden face removal and clipping.

For example if a near polygon is drawn into the display memory the z–value of each pixel will be low. If another polygon is to be drawn which cuts through the original polygon, as each of the pixels are drawn, the z–values of the new polygon are compared with the z–values in the display memory and if the new pixel is closer (i.e. has a smaller z) the pixel from second polygon is drawn into the memory (including z–value) otherwise the pixel from the original polygon remains. Therefore, once a polygon has been drawn it may be forgotten.

When image composition is to be used, there are multiple overlapping display memories which contain a part of the scene. This means that when the data is read from the memory to be displayed instead of one stream of pixels there are multiple streams of pixels. It therefore becomes necessary to decide which of the stream to choose to display. This is done on a pixel–by–pixel basis by comparing the pixel with the lowest z component. As a result the comparison of z–values must occur at pixel display rates.

Objects in the virtual world will appear to change at different rates. The apparent movement of the object due to a rotation of the user is handled automatically by the address recalculation pipeline which implements delayed viewport mapping. Other changes in the way the object is perceived are due to the object's own animation and due to translational movement of the user in the virtual world. It is possible to classify further kinds of changes in the appearance of the object into those involving its displayed size and those involving its displayed position. Objects should appear larger when the user is closer to them and smaller when farther away. Relative side–to–side movement will cause an apparent change in the position in which the object is displayed. Of course any animation of the object itself can lead to both these effects.

The effects of user translation can be quantified using simple geometric analysis. Using this approach it is possible to calculate the amount of time for which the current displayed image of the object will remain valid, and hence determine the required rate of update of the object's displayed image.

Prioritised rendering takes advantage of the fact that not all objects need to be updated at the same rate. By sorting the objects into a priority order based on their required update rates, the most important changes in object appearance can be rendered first, thus guaranteeing that if perchance there is insufficient rendering capacity the best possible result has been achieved. Even in such an unfortunate case the latency of response to head orientation changes is not affected since that is handled by the address recalculation pipeline 100.

This however is not the main advantage of prioritised rendering. Its most dramatic property is that the overall rendering required is reduced significantly. It can be shown that an order of magnitude reduction in required rendering is not unreasonable due to this approach. This occurs because the average required update rate for all objects in the virtual world is much less than the display update rate. By balancing the rendering load among multiple renderers updating multiple display memories at various update rates and composing the final image for display, the combination of prioritised rendering, image composition, and the delayed viewport mapping provided by the address recalculation pipeline can outperform conventional virtual reality implementations.

In employing the image overlaying or image composition technique, multiple display memories are required rather than just one display memory (or two for double buffering). Different sections of the visible scene may be drawn into separate display memories then overlayed to form a final scene. In many implementations each display memory has a private rendering engine.

As pixels are being fetched from the display memory to be sent to the output device, all the display memories are simultaneously fetched from the same location.

Next the z–value associated with each pixel is compared with the z–value of the pixels from the same location in the other display memories. The pixel with the smallest z–value is sent to the output device.

In a conventional graphics system image composition uses a factor n redundancy to provide a factor n increase in performance. A side effect of image composition is that each of the display memories may be updated individually and at different rates. In a virtual reality system using image composition alone this side effect is generally of no use as all of the images in the display memories are rendered with the same fixed viewport mapping so when the viewport changes due to a user head rotation all of the images have an incorrect viewport mapping and need re–rendering. A factor n increase in speed is all that may be achieved.

Using the address recalculation pipeline it is possible to make effective use of this side effect of image composition to achieve in certain cases much better than factor n improvement for n display memories in a virtual reality display environment. This is because the images in the display memory of a graphics system with an address recalculation pipeline do not necessarily become invalid when the user's head orientation changes, thus the length of time an image in display memory is valid only loosely depends on the orientation (for a stereo view). For example a non interactive background may never require re–rendering and may thus be pre–rendered with greater detail using a high quality rendering technique and a complex model.

Using the address recalculation pipeline 100 allows scenes to be rendered which are largely independent of the user's head orientation. When image composition is combined with the address recalculation pipeline it is possible to render different parts of a scene at different rates.

The orientation independent sections of a static scene that change the most tend to occur during user translations. When the user is stationary within the scene, the renderers must only maintain stereoscopy (which is a form of translation) and animate objects which are changing themselves or change as a result of interaction.

Priority rendering is demand driven rendering. An object is not redrawn until its image within the display memory has changed by a predetermined threshold. In a conventional system this strategy would not be effective as almost any head rotations would cause considerable changes to the image in display memory and the system would have to re-render everything. The images stored in the display memory of a graphics system with the address recalculation pipeline are to a great extent independent of user head orientation which means the renderer does not have to redraw the entire scene for head rotations.

The threshold for determining when an object has changed by more than a tolerable amount is determined by the designer of the virtual world and may typically be based on several factors. Usually this threshold is in the form of an angle (θ_t) which defines the minimum feature size of the world. Ideally this angle would be less than the minimum feature size the human eye can detect, approximately one third of an arc minute, however in reality this is impractical. If anti-aliasing of the image in display memory is not used a more sensible threshold may be the inter-pixel spacing in the display memory and if no hardware anti-aliasing is used at all, the pixel spacing in the head set worn by the user may be used. Priority rendering attempts to keep the image in display memory accurate to within (θ_t) at the highest possible update rate.

In order to compute the image changes for an object contained within the virtual world we compute how much the object would have changed if it were static and then add an animation component unique to the object as required.

Consider what happens to the display memory image of a static object as the user translates relative to the object. The relative location of the image changes and the image itself may change in size. The rendering strategy must compensate for image changes within display memory. It is possible to predict when these changes occur by observing certain features of a sphere which encapsulates the object.

User translations cause objects to move within the display memory. In order to keep the scene accurate to within θ_t we need to know how long the image of the object will remain valid at the user's current relative-speed. This time is known as the object's translational validity period ($\tau_{translation}$). Relative-speed is used to compute the object's validity period rather than relative velocity as the resulting world would have several objects caught in slow display memories if the user changes direction significantly. This would result in large temporal errors in the locations of several objects. The relative-speed must include a component for the eyes' speed relative to the centre of rotation of the head if the user's head is rotating. Figure 15 illustrates how the translational validity period $\tau_{translation}$ is calculated for a particular object, i. In effect, the translational validity period is the time period required for the centre of the object to trace an angle θ_t across the image presented to the user, and can be calculated according to:

$$\tau_{translation}\ (i) = \frac{\text{distance (i)} * \sqrt{(2 * (1 - \cos(\theta_t)))}}{\text{relative\_speed(i)}}$$

where distance (i) is the computed distance between the object, i, and the user; and relative-speed (i) is the computed relative-speed as between the object and the user.

As the user moves towards or away from an object the size of the image of the object changes. We must compute the time that the size of the object is valid and re-render the object when its image size has changed by the predetermined threshold θ_t. Again we use the speed of the object relative to the user for our computations for the same reasons as before. The period for which the size of the image is valid is know as the object's size validity time (τ_{size}). Figure 16 illustrates how the size validity time τ_{size} is calculated for a particular object, i. In effect, the size validity time is the time period required for the perceived angular distance between the centre and outer periphery of the object to diminish by an angular amount of θ_t. As shown in Figure 16, the size validity period can be calculated according to:

$$\tau_{size}\ (i) = \frac{\text{distance(i)} - \text{radius(i)}/\sin(\theta_t + \text{A}\ \sin(\text{radius(i)}/\text{distance(i)}))}{\text{relative\_speed(i)}}$$

where radius (i) is the computed distance between the centre of object i and the outer periphery thereof. Note that as the image size change by θ_t there may be several aliases of the object; these have been ignored.

The last factor we consider is any requirement of animation by the object itself. For example a bird flapping its wings requires more updating than a static object like a stationary rock. The period of update for a specific object must be tagged to the object within the database and is defined by the virtual world designer. The period of the current frame of animation for a particular object is known as the object's animation validity time.

$$\tau_{animation} = \text{user\_defined}$$

Many other factors may be considered relevant for a highly accurate representation of the scene. For example, object rotation corresponding to user translation is because either size, translational or animation changes tend to dominate the required update rate.

Finally it is possible to determine the overall object validity period. This period indicates the amount of time available until the next update of the object is required. This period also indicates the latency for a particular object, however, by definition the error in position of the object is less than θ_t. The overall object validity period $\tau_{overall}$ is defined as the smallest of the translational, size and animation periods. Obviously if $\tau_{overall}$ is less than the period of maximum update rate, $\tau_{overall}$ is assigned the period of the maximum frame rate. This period defines the object's priority (i.e. smaller period means higher priority) and the rendering power devoted to a particular object is based on this priority.

Accelerations have not been considered thus far, only relative–speed. This may result in latency when accelerating as an object's computed validity period may not accurately reflect the actual validity period. Including acceleration into period computations is possible however the computation is made unnecessarily complex as high accelerations within a virtual world are limited as the sense of heavy acceleration may result in a form of motion sickness known as vection.

The above is based on being able to render all objects completely independently, this would require a pair of display memories per object (for a stereo view). As display memory pricing tends to dominate the overall system cost, providing one display memory per object is obviously impractical. An alternative is to have a limited number of display memories 16 with fixed update periods and attempt to allocate objects by choosing the memory 16 with the highest update period, which is less than the validity period of an object, as the target display memory 16 for that particular object.

Several strategies for dividing the overall system into a set of display memory update rates are possible and the optimal technique will ultimately depend on the nature of the virtual world. For prototype implementations of the priority rendering a set of update rates starting at the highest swap rate (for example 60 Hz) were chosen. All other swap rates are some exponential harmonic of the top rate. The display memory update swap strategy is depicted in Figure 17 which shows the update sequence for four memory layers (memory 0 to memory 3). The memory which is updated most often (memory 0) should contain the objects with the smallest overall validity period, whilst the memory updated least often (memory 3) should contain objects with the largest overall validity period. As can be seen from Figure 17, memory 0 is updated twice as often as memory 1; memory 1 is updated twice as often as memory 2; and memory 2 is updated twice as often as memory 3. Using this technique it is also possible to swap an object from a low update rate into any of the higher update rates, or vice versa, if required.

The fastest display memory updates at 60 frames per second and all of the other display memories update at some exponential harmonic of the top rate. The main reason for basing the updating rates on harmonics is so objects may be swapped from a slow update rate to a faster update rate, if required, by swapping from one display memory to another. The promotion or demotion of an object from one update rate to another occurs on the crossing of the harmonics, since if objects are not swapped on harmonic crossings an object may not be represented, or represented twice, for a short period of time. This choice of exponential harmonics may not lead to maximum rendering efficiency (the number of times the object needs to be updated compared with the number of times the object is updated) and rendering loads across all display memories may not be distributed evenly. However the optimal configuration is based heavily on the nature of the virtual scene, the rendering power available and the desired overload strategy.

The rendering hardware may have more display memories available than the virtual world requires for high efficiency. In this event, multiple renderers and display memories may be assigned to the one update rate thus devoting more hardware resources to a particular update rate, helping to balance the load.

The previous computations do not take into account stereoscopy. Fortunately the closest objects are most likely to be in high-speed buffers and it is these objects that are most affected by stereo updates. It may be possible to include a period factor which considers how far the head may rotate within a set period of time, however this is deemed unnecessary as some latency to stereoscopy is available.

Database distribution for priority rendering needs to be considered. In general there are two methods of generating primitives for display.

(1) Immediate Mode, as shown in Figure 26. In immediate mode the application program generates new primitives each display frame. This form of operation is not really suited to the priority rendering approach. No assumptions are made by the graphics subsystem as to the inter-frame similarities of a sequence of display frames. If desired, the application may generate different primitives each and every frame. To take advantage of priority rendering the application program must be aware of the details of the underlying rendering system and is responsible for determining object validity, object assignment and load management.

(2) Retained Mode, as shown in Figure 27. This mode of operation is more suited to priority rendering. In retained mode the application creates a display list 250 and then the graphics subsystem traverses this list to generate an image. Priority rendering may be performed with little help from the application. Each

rendering engine 252 stores a complete copy of the displ: . list so as to minimise the communications overheads when objects transfer between rendering engines. The application communicates with the priority rendering manager 254 which broadcasts changes to the display list made by the application. The priority rendering manager 254 also has a copy of the display list which contains extra information necessary to determine object validity and estimated rendering time. The priority rendering manager uses this information to determine which of the renderers are responsible for the display of various objects and it is also responsible for managing the load on various renderers.

In applications which use a hierarchical display list, an object may in fact be part or all of an entire subtree of the hierarchy. The grouping of elements in the hierarchy to form objects is the responsibility of the applications programmer. At this time $\tau_{animation}$ is also defined. For example a car description may contain wheels at a lower level of the hierarchy. It is up to the applications programmer to decide whether the wheels are separate objects to the car object, or whether they are all treated as a single object.

Image composition architectures work by distributing the display database across several renderers and rendering each sublist in parallel. The load balancer is responsible for ensuring even processor utilisation to minimise the chance of a single rendering engine delaying the entire rendering system by a full frame. When priority rendering is introduced the requirements change somewhat. The load manager 254 is responsible for ensuring that all the processors are evenly loaded and that objects are assigned to renderers 252 with appropriate update rates. In the priority rendering environment the load manager has the advantage of being able to reduce the rendering load on a per-object basis rather than a per-scene basis. In a conventional image composition architecture, when the rendering load is greater than the rendering performance available the only choice which maintains image quality available may be to reduce the frame rate. With priority rendering it is possible to reduce the update rate of a subset of objects, reducing the rendering load to a level lower than the available rendering performance.

Once the validity periods for all of the objects in a scene has been determined, they must be assigned to renderers with appropriate update rates. The algorithm responsible for object assignment faces two problems. The first problem arises when an individual renderer is unable to fulfil its rendering requirements in the allotted time. The second problem occurs when the overall rendering load required is greater than the rendering performance available.

The first problem arises from the difficulty in predicting rendering times. Although all efforts may be taken not to overrun a renderer, the time taken to render a fixed number of primitives may vary significantly between updates. Several factors may affect the time taken to render a set of objects. Some of these factors include:

(a) clipping. The clipping load faced by the geometry engine may vary greatly depending on the positions of various objects in the scene. This may be particularly relevant for renderers with high update rates. It is conceivable that the renderer with the highest update rate has a only a very small number of objects assigned to it. If these objects are in the middle of one of the display memory faces there may be no clipping required. Should these objects move to the corner of the cube, all objects may need to be clipped against three edges, significantly increasing the clipping load.

(b) object distance. As the distance to an object increases, it is likely to become smaller and the number of pixels required to rasterise the object will go down. In many rasterisation architectures the time of rasterisation is directly related to the number of display memory accesses. All may not be lost however. It is unlikely that the size of an object will change significantly between updates (less than θ_t ideally), so the time take to rasterise an object on its last update may be a good indication of its current rasterisation time. Also it may be possible to predict the change in rasterisation requirements. For example, the number of pixels required to rasterise an individual polygon in an object is likely to be inversely proportional to the square of its distance.

(c) rasterisation order. The order of rasterisation may affect the time taken to rasterise pixels. If primitives are ordered front to back, the last objects to be rendered are less likely to be visible hence the number of display memory accesses will be reduced from two to one per pixel (only a z-buffer read is required, rather than a z-buffer read and an RGBz write). If primitives happen to be ordered back to front, the rasterisation of all primitives will require two display memory accesses per pixel (a z-buffer read and an RGBz write). This problem may not be as relevant if using newer display memories which perform z-comparisons on-chip such as Deering's 3DRAM, as discussed in Deering M., Schlapp S. and Lavelle M., 1994, "FBRAM: A new form of Memory Optimized for 3D Graphics", *Proceedings of SIGGRAPH '94 (Orlando, FL., 24-29 July, 1994), In Computer Graphics, Annual Conference Series 1994,* pp. 167-174.

(d) spatial level-of-detail. Priority rendering is a temporal level-of-detail technique but it is compatible with spatial level-of-detail switching. Altering the spatial level-of-detail is a known method for reducing rendering time due to an object's simplified geometry and reduced chance of containing single pixel polygons. When an object undergoes a spatial level-of-detail change, its rendering time may change significantly and unpredictably.

The time taken to render a number of primitives may depend on several factors and as such it may be difficult to predict accurately.

Renderer overrun is the problem faced when an individual renderer is unable to completely render all of its objects in the time it has been allotted. Renderer overrun may be avoided by predicting the number of objects a renderer can render in a fixed time slot and allocating the left over objects to a renderer with a lower update rate. In the event of renderer overrun, two strategies are possible. The simplest strategy is to perform a double buffer swap regardless of rendering completion. In general, this would be highly undesirable as some objects may flicker between partial and full completion. Other objects may disappear all together.

The other strategy is to insert an additional frame. That is, no renderer may perform a double buffer swap unless all renderers scheduled to swap at a particular time have completed their image, effectively delaying the swap by one frame. This means all renderers with a lower update rate than the renderer which caused the delay have an additional frame to complete their images. Thus reducing the chance of a renderer with a lower update rate overrunning. Renderer overrun is depicted in Figure 28 where at time 260 Renderer 2 is unable to finish in the allotted time. An additional frame 262 is inserted and no buffer swaps occur until all renderers are scheduled to swap complete their images.

If the renderer that overruns on a scheduled double buffer swap is the renderer with the lowest update rate scheduled to swap at a particular time, another problem arises. The problem is that the cause of the overload (too many objects updating too frequently) cannot be moved to a renderer with a lower update rate. This means there is likely to be a ghost overrun when the renderer that caused the overrun is scheduled to swap again. The second overrun is called a ghost overrun as illustrated in Figure 29 a ghost frame 264 is inserted and Renderers 0, 1 and 2 are unlikely to dissipate the cause of the overload. Moving objects to renderers with higher update rates only increases the overall load potentially adding to the overload situation or increasing the number of potential overruns.

It is expected that renderer overload leading to renderer overrun is the result of bad prediction rather than significant changes in geometry processing or rasterisation. As such ghosting is expected. The time taken to render an image is not expected to vary from update to update as much as in conventional rendering systems. The reasoning behind this logic is that the rendering load in a conventional rendering system is dependent on the viewport orientation. The rendering load depends on the number of objects culled by the trivial rejection stage, which rejects objects completely outside the viewport. If the user is looking at complex section of the scene the rendering load is likely to be quite high. When the viewport moves to a barren section of the scene, the load is likely to drop significantly. In a priority rendering system no primitives undergo trivial rejection so the update to update load is likely to be more even. The exception is when some human factors are considered, as discussed hereinafter. In some circumstances objects assigned to renderers with high update rates may undergo a form of trivial rejection. This may lead to a condition where the rendering load is dependent on the orientation of the viewport and highly variable.

An object assignment strategy is used to avoid renderer overrun, to utilise fully all renderers and to minimise the number of objects assigned to renderers with inadequate update rates. The following object assignment strategy is not the only strategy possible, however it fulfils the above mentioned requirements.

Clearly if the rendering load computed from the τ_{valid} (the time a rendered object is valid) of individual objects is greater than the rendering performance available, some compromises must be made. If there is insufficient rendering performance the user cannot complain about loss of animation quality (rotational interactivity is not affected).

Ideally an object should be assigned to a renderer with at least the update rate that the object requires. If this were the assignment strategy, it is likely that renderers would become overloaded and cause many renderer overruns. Another assignment strategy is to sort a subset of the objects by τ_{valid} after each double buffer swap. The sublist contains all objects not already assigned to a renderer with a lower update rate. The fastest renderer is selected first and objects are assigned to the renderer from the ordered list. The rendering load of each object is estimated from its previous rendering time. A running total of the predicted load on a renderer is kept and when it reaches a high (threshold) percentage of the renderer's peak performance the next renderer is selected. This process continues until the last renderer is selected. All remaining objects are assigned to the last renderer. When too many renderer overruns occur, a new set of harmonics may be selected and applied when the renderer with the lowest update rate is due to be updated. Pseudo-code for the assignment strategy is given below. The technique may be bootstrapped by initially assuming a uniform rendering load per object.

```
// This routine is executed immediately after a double buffer swap.
// Not all renderers may have performed a double buffer swap so max_renderer
// is assigned the number of the render with the lowest update rate which
// performed a double buffer swap.

        object_list = remaining_objects;
        compute_$\tau_{valid}$ (object_list);
        sort_objects (object_list);                    // objects are sorted by $\tau_{valid}$
        for (renderer = 0; renderer < max_renderer; renderer ++) {
              render_load = 0;
              while (render_load * update_rate (renderer) < threshold*performance) {
                    assign_object (renderer, first (object_list));
                    render_load += last_rendering_time (first (object_list));
                    remove_first_object (object_list);
              }
        }
        renderer = max_renderer;
        while (not_empty (object_list)) {
              assign_object (renderer, first (object_list));
              remove_first_object (object_list);
        }
```

The choice of threshold which determines when a renderer has enough objects is quite important, it provides a buffer zone for errors in prediction but it also determines the average renderer utilisation. Increasing the threshold increases processor utilisation (by decreasing idle time) but may lead to a greater risk of renderer overrun.

The above mentioned strategy has several advantages designed to give graceful performance degradation under overload conditions while avoiding renderer overruns. When the system becomes overloaded, several objects may still be rendered with a high update rate (and rotational interactivity is never lost). The system may remain highly interactive under heavily overloaded conditions. The interactive performance of less important objects (those with a high τ_{valid}) is sacrificed at the expense of more important objects (those with a low τ_{valid}).

When implemented efficiently, with a small enough object list, the execution time of the above pseudo-code may be hidden by the time spent clearing the display memory's RGBz values after each double buffer swap. During this period the rendering engines are unable to rasterise any primitives.

In order to increase the level of realism experienced by a user, it is desirable to display a stereoscopic image of the virtual world. This may be achieved by using two identical address recalculation pipeline (ARP) systems 100. Clearly when a user's head rotates the centre of rotation is not located at the user's eyes. This means the eyes undergo some form of translation during a head rotation.

There is a physical bound on the rate at which a human head may rotate. This means there is maximal translation speed of the eyes relative to the centre of rotation. To handle stereoscopy, the centre of projection of each of the ARP systems is moved to the centre of each of the user's eyes, and the relative speed of all objects in the world is increased by the maximum translation speed of the centre of projection. For practical purposes this is in the order of 0.5 metres per second. In many environments, the user's translational speed will be high enough to make the increase negligible. Fortunately the closest objects are most likely be assigned to renderers with high update rates and it is these objects that are most affected by stereoscopy requirements.

Replicating the complete system for stereoscopy may at first seem unnecessary; distant objects are unlikely to require any significant correction for stereoscopy and therefore the images of distant objects generated by the renderers for each eye are likely to be the same. This may be true however a problem can arise when asymmetric wide angle viewing lenses are used in the head-mounted display (HMD). When asymmetric lenses are used, the ARPs for each eye may not address the same pixel location at the same time. Only when symmetric lenses are used in the HMD will the fetch locations of the ARPs be the same.

Should a single display memory contain distant objects for both eyes, the memory would have to be read twice per pixel display, once for ARP_{left} and again for ARP_{right}. Display memory fetch time is already one of the constraints on the output resolution, so halving the fetch time would require memories with half the access time or require replication of the memory. In effect this means that when the system is replicated for stereoscopy, only some of the rendering engines are redundant rather than the rendering engine/display memory combinations.

It is conceivable that some large objects could be decomposed into smaller objects and should these smaller objects be handled individually they might not all be rendered at the same rate. For example, a fractal terrain may contain several thousand polygons yet most of these polygons are some distance from the user. Only a small number of polygons near the user would be updated at a high rate.

A problem arises in decomposing objects which contain shared vertices, namely an opening of size θ_t could appear in an object where part of the object is contained in one renderer with the location of vertices computed at time t, as shown in Figure 30, while other parts of the object are in another renderer with the location of vertices computed at t+1, as shown in Figure 31, where the polygons are rendered at different rates and Polygon 2 is re-rendered while Polygon 1 is not and common vertices (C and D) may have more than 1 location, resulting in an opening in the surface. This problem is caused by having the same vertex split across two renderers, and having the location of the vertex computed at different times for the different renderers.

The time $\tau_{translation}$ tends to dominate τ_{valid}, and this indicates that the radius of an object is much less important than the distance to the object. As such, the τ_{valid} computation may be performed on vertices rather than completed objects.

One possible technique to divide objects amongst renderers with different update rates is to keep the location of vertices common between renderers and update the location of vertices based on τ_{valid} of the vertex rather than the τ_{valid} of the objects that may contain the vertex. An object is updated at the rate of the fastest vertex in the object. The result is that polygons which share vertices with polygons with a lower update rate will be sheared so the vertices meet up with those vertices contained in objects rendered at a lower update rate. This is illustrated in Figure 32 where Polygon 2 is re-rendered with two new vertices, E(t+1) and F(t+1), and two old vertices C(t) and D(t) and the surface remains closed.

This technique is some what ad-hoc and may result in undesirable artefacts for small or distant polygons. Potentially, polygons rendered with new and old vertices could cross in the middle. The technique does however mean that large objects may be distributed across several renderers. In some simulations the ground plane may represent the majority of the polygons in the world, so being able to perform priority rendering on a single large object is highly desirable.

The priority rendering method described above assigns an object to a renderer based on the period of time for which the image of the object remains unchanged. Essentially the display surface completely encapsulates the user's head. All primitives are rasterised, when the trivial reject stage of the rendering pipeline is not implemented.

When human factors are considered it becomes possible to perform trivial rejection on objects which the user has no chance of seeing. For example it is pointless rendering objects in the renderer with the highest update rate which are directly behind the user; the user simply is not capable of rotating his head fast enough to actually see these objects before the next update from the renderer.

Assuming the user's head may rotate at approximately one revolution per second and the maximum update rate is 60 frames per second, the user's head may rotate 6° between updates. Allowing up to 3 frames of image generation latency and assuming the user's field of view in the HMD is 90°, objects which are more than 63° (90°/2 + 3*6°) away from the user's view direction do not require rendering in the 60 fps renderer. If the next renderer operates at 30 fps, objects more than 81° (90°/2 + 3*12°) from the user's view direction need not be rendered, and so on. Figure 33 graphically depicts the culling for a series of renderers. Increased efficiency against maximum rotation speed for an experimental environment, is discussed hereinafter.

Performing trivial rejection on primitives outside the possible viewing angle requires one dot product computation and one comparison for each vertex of an object but reduces the number of objects which undergo geometric transformations, clipping and rasterisation thus reducing the overall rendering load.

The actual upper bound on user head rotation speed is still the subject of investigation.

When the Relative Object Rendering Load (RORL), defined below, is high, most of the objects are rendered by the renderers with a high update rate. In this case the human factors trivial rejection works similarly to the trivial rejection stage of a conventional system which means the performance with increasing RORL will tend to approach that of a conventional system rather than having worse characteristics than a conventional system.

One experimental environment c: virtual world application which has been investigated is a walk through of a forest. This simulation was performed in order to determine the rendering load on various display memories with various update rates.

The virtual world under investigation contained one thousand trees, each tree bounded by a sphere of radius five metres which implies a maximum tree height of ten metres. The actual number of polygons contained within each tree is arbitrary as only the object rendering load is considered, the number of polygons per tree eventually being determined by the real rendering power. The trees are randomly placed in a circular forest at a density such that the leaves of the trees may form a continuous canopy. The resulting forest has a radius of one hundred and fifty metres.

The simulation investigates the object rendering loads on various display memories with different update rates. The simulation is conducted as a walk through the world from one side to the other at one metre per second (approximate walking speed), passing through the centre of the virtual world. This enables statistics on rendering loads to be determined for circumstances ranging from being completely surrounded by the objects to being at the edge of the objects. The chosen allowable error θ_t was the smallest inter-pixel spacing between he smallest pixels in the display memory. For a system with a display memory resolution of 512 by 512 pixels per face, this means the smallest distance between any two pixels is approximately six arc minutes. This is an order of magnitude higher than the resolution of the human eye.

All of the comparisons are based on the number of objects that must be redrawn by the rendering system to maintain approximately the same illusion with an effective update rate of 60 frames per second. A comparison is made of the number of objects a system with the address recalculation pipeline 100 must redraw against the number of objects a system without the pipeline must redraw for the entire length of the simulation (both systems are assumed to have multiple display memories and multiple renders). The Relative Object Rendering Load (RORL) is a percentage measurement of this ratio.

$$\text{RORL} = \frac{\text{Total number of object updates (with pipeline)}}{\text{Total number of object updates (without pipeline)}} * 100\%$$

With the address recalculation pipeline 100 for the simulation described above the RORL was determined to be 15%. That is the system with the pipeline only had to redraw 15 objects for every 100 objects the system without the pipeline had to redraw for a similar illusion. If the minimum feature size (maximum error size) θ_t is made larger, the RORL reduces further. When θ_t is increased to be the smallest arc distance between the largest pixels in display memory the RORL is less than 8% of the object rendering load of a conventional system. Figure 18 depicts a graphical representation of the relationship between the object rendering load (relative to a conventional system) and minimum feature size.

As can be ascertained from the above, the combination of the address recalculation pipeline with image composition and priority rendering can reduce the total object rendering load to 15% of the equivalent object rendering load without the new hardware. The main reason for this significant saving is depicted in Figure 19. Of the total number of object updates required for the walk through, nearly 40% of them were assigned to display memory 3 which is swapping at 7.5 frames per second. So even though display memory 0 is swapping 8 times faster than display memory 3, it is only doing one quarter the work display memory 3 is doing. This means memory 3 is updating (40%/10%*8) = 32 times the number of objects display memory 0 is updating (at an eight the rate).

Providing a stereo view of the world is highly desirable within a head-mounted graphics display system to help with the sense of presence within the virtual world. With an address recalculation pipeline the display memories are not actually centred around the point of rotation of the user's head, rather they are centred around the user's eyes. This means when the user's head rotates while the user is stationary a small amount of translation occurs. This implied the need to re-render some objects which are affected by the translation caused by the head rotation. Although the speed at which the eyes translate during a head rotation must be included into the priority computation for $\tau_{translation}$ and τ_{size} it is interesting to note the total number of objects that become invalid to head rotations of various angles. Figure 20 shows a graphically represented example of how many objects become invalid for a particular head rotation. The upper line 160 is for θ_t = 6 arc minutes (corresponding to the smallest pixel in the display memory) while the lower line 162 is for θ_t = 13 arc minutes (corresponding to the largest pixel in display memory). From this graph we see that head rotations smaller than 45° require few objects to be updated. The graph of Figure 20 was generated using data obtained whilst situated in the middle of the above mentioned virtual scene.

Translations within a virtual world are animated, hence the rendering engine is involved in the process. To allow translational freedom a powerful rendering engine is required. A series of Intel i860 microprocessors, for example, are considered suitable for rendering. Each depth for each eye can be allocated an i860 to write an image into the display memory. Three depths are to be implemented for each eye resulting in a total of six i860s within the rendering engine.

In practice, a high performance, multiprocessor host computer will generate the scene to be displayed by generating all of the required polygons to describe the scene, acting as a polygon server. The i860s then scan convert the polygons. Using this form of hardware it is possible for the display memory to be double buffered with two 16-bit colour fields and a 16-bit z-buffer field for each pixel. One way in which the prioritised rendering method can be implemented is to provide a software function in the host computer which stores, along with the polygon structural information, an indication of the priority of the polygon based on the overall validity period for the object containing the polygon. It is then a relatively simple matter to provide the software for the i860 based rendering engines with functions for selecting only polygons with an appropriate stored priority value to generate image data for the various display memories. The software for the polygon server host computer can be synchronised with the update periods of the rendering engines so that the priority value for a polygon is only altered by the host computer at a harmonic crossing for the update periods, as discussed hereinabove. An otherwise conventional polygon server host computer and rendering engines can be utilised.

The address recalculation pipeline 100 performs a remapping of the image in display memory to form an output image in real time based on the wide angle viewing lenses and the user head orientation. This remapping is performed one pixel at a time by the hardware in the address recalculation pipeline. The remapping occurs by sampling the image contained within the display memory and as with any form of discrete sampling, aliasing occurs. Even if the image in the display memory is anti-aliased and rendered with a high quality rendering technique, the hardware sampling occurring will cause aliasing in the final image. The aliases introduced by the address recalculation pipeline cannot be corrected with software in the rendering process. Any pipeline anti-aliasing must occur in the hardware.

Providing no hardware anti-aliasing is always an option and in fact simplifies the hardware significantly, although the display image quality can suffer quite severely. For example, images become sensitive to noise from the head tracker. When the intersection computation from the pipeline for a given pixel in display memory is close to the pixel's area boundary, a very small change in the tracked position of the user's head caused by tracker noise could cause the output pixel to flicker between adjacent display memory pixels. The flickering will be most noticeable at the boundaries of polygons and on straight lines. Even stationary objects will have noticeable artefacts. Straight lines appear broken and the staircasing of angle lines become significant.

The anti-aliasing strategy chosen for the present display architecture is a linear interpolation filter using redundant addressing bits from the intersection computation.

With the current architecture 4-bits in each direction are used for the interpolation filter, as shown in Figure 21. A linear interpolation filter provides an adequate trade-off between system expense and filter quality.

In order to perform linear interpolation the four pixels surrounding the point of intersection must be fetched simultaneously. The interleaving mechanism for fetching the four adjacent pixels is discussed later.

Referring to Figure 21, intersection point 200 indicates the result of the intersection computation from the pipeline representing the point of intersection of the vector 120 (see Figure 8) with the encapsulating display surface. As can be seen from the drawing, the intersection point 200 is not aligned precisely with any one individual pixel from the encapsulating surface display memory, but is disposed between four adjacent pixels 202, 204, 206 and 208 (pixel 1, pixel 2, pixel 3 and pixel 4). The linear interpolation filter used herein for anti-aliasing in effect takes a weighted average of the colour intensities of the adjacent pixels 202, 204, 206 and 208 on the basis of the distance from the intersection point 200 to each of the adjacent display memory pixels. The three colour streams of the four adjacent pixels 202, 204, 206, 208 are separated and managed independently. The four adjacent pixels are paired off such that the two Y values are the same. That is pixel 1 (202) is paired with pixel 2 (204) and pixel 3 (206) is paired with pixel 4 (208). Within each pair of pixels, the colour component of the first pixel (202 or 206) is concatenated with the second pixel (204 or 208) colour value which is then concatenated with the (x) redundant bits from the intersection computation. These form two addresses into two independent look-up tables which contains interpolated data corresponding to the colour values for the individual pixels of each pair and the relative positioning of the intersection point with respect to the pixels of each pair along the x-axis. The output of the look-up tables corresponds to intermediate pixels 210, 212 which are the colour interpolation of the two pixel pairs. The intermediate pixel values are concatenated with each other and the (y) redundant bits from the intersection computation, to form an address into another look-up table which interpolates along the y-axis direction. The output of this table is the final colour interpolated value of the four adjacent pixels 202, 204, 206, 208.

Figure 22 depicts a block diagram of the hardware structure for a linear interpolation filter 220 which operates as discussed above. The filter 220 illustrated represents only a single colour value (red) of the three colour values used for each pixel. The interpolation of the pixel colour values along the x-axis direction is performed by look-up tables 222 and 224 which each receive as inputs the colour values for the respective adjacent pixels and a 4-bit positioning address which represents positioning of the intersection point 200 along the x-axis between the adjacent pixels. For example, the look-up table may be structured such that if the 4-bit positioning address is 0000, the output value from table 222 is the same as the colour value for pixel 1. This would correspond to the case where the intersection point 200 is aligned with pixel 1 and pixel 3 (202, 206) in the x-direction. Similarly, if the positioning address has a 1111 value, the output from table 222 may be the same as the colour value for pixel 3. Where the positioning address is an intermediate value, the look-up table can be structured so as to return a value corresponding to, for example:

$$\text{Output} = \frac{(\text{input x} * \text{input 1} + (16 - \text{input x}) * \text{input 2})}{16}$$

where input 1 is the colour value for pixel 1 (table 222);
input 2 is the colour value for pixel 2;
input x is the 4-bit positioning address; and
output is the value stored in look-up table 222 corresponding to the address formed by concatenating input 1, input 2, input x.

A similar procedure is then carried out by the look-up table 226 using the outputs from tables 222 and 224 as inputs, as well as a 4-bit y-axis positioning address. The 4-bit positioning addresses used as inputs to the look-up tables 222, 224 and 226 of the interpolation filter 220 can be obtained from the four least significant bits of the x and y components of the vector 120 generated by the matrix multiplier 116 of the pipeline 100 (see Figure 8).

The number of bits per pixel colour component obviously has a great influence on the size of the look-up tables. When pixels in display memory are stored in a

24-bit colour format, each of the three look-up tables for one colour stream is 1 Mbyte. The overall anti-aliasing strategy requires 9 Mbytes. As the pixel interpolation occurs after image composition this figure of 9 Mbytes is independent of the number of display memories. While this figure may seem high for an anti-aliasing strategy, a complete system with six display memories would have over 160 Mbytes of display memory and an extra 9 Mbytes is easily justified. When 16-bits per pixel are used for colour instead of 24-bits, the overall anti-aliasing strategy requires a little over half a megabyte. The intermediate pixel values are registered hence forming another pipeline stage. The anti-alias look-up table should also be static memory operating at video display rates since it forms part of the pipeline.

The display memory in the address recalculation pipeline system is generally one of the most expensive components in the system, so the organisation of the memory is important, moreover any inefficiencies are multiplied by the use of image composition.

Conventional graphics systems generally use low-cost dynamic RAM in a fast page mode or video RAM which has internal shift registers to achieve the high output bandwidth required by raster display systems. The high output bandwidth is achieved in both cases as an artefact of the sequential nature of the display memory accesses. Pixel addresses created by an address recalculation pipeline are not sequential and while there may be some locality in consecutive pixel addresses, these addresses are considered to be random. The display memory must be able to perform random accesses at raster display rates. This means that with current technology the display memory has to be implemented with expensive high-speed static RAM. Interleaving of the video memory cannot be used to increase the effective video memory access rate which means the random access time of the static RAM must be less than the pixel display period, hence making the display memory quite expensive.

The display memory is organised into 6 faces for a cuboid encapsulating surface, where each face has a resolution of 512 by 512 pixels, the faces being illustrated and arbitrarily labelled Face 0 to Face 5 in Figure 23.

In order to perform a linear interpolation anti-aliasing strategy, adjacent pixels within display memory are accessed simultaneously. The access mechanism used in the prototype system begins by first dividing the display memory into 6 independently addressable segments and then arranging them in such a manner that with appropriate logic, four of the six addressed memory accesses retrieve are the required adjacent pixels. The independently addressable segments are labelled A, B, C, D, E and F. The address range of the segments are grouped but different sections of the address range of each segment are grouped differently, for example the address bits 0-7 of segment A are grouped differently to bits 8-17 of segment A. Note that the addressing figures given are for a system with a face resolution of 512 by 512 pixels. The exact groupings are shown in Figure 23.

Face intersection computations from the address recalculation pipeline give a 3-bit face number f, and two 9-bit face coordinates, i and j. Lets first consider the i face coordinate. We arrange the memory such that for a given address the pixel in segment A is always to the left of the pixel in segment B. The same is also true for segments CD and EF. For a given intersection, if the required order is AB, segments A and B may be accessed with the same addresses, however if the required order is BA, segment A must access the pixel one location further into the memory than segment B. The required order AB or BA is given by the least significant bit of i. The display memory lines are 512 pixels wide which means there must be 256 AB pixel pairs. Thus A and B line position coordinates must be 8-bits wide. For any i, the B pixel low order address bits are computed by shifting i right by 1-bit, i.e. B(0-7) = i>>1. The A pixel low order addresses are computed by adding one to i then shifting the result right 1-bit i.e. A(0-7) = (i+1)>>1. As A must also be adjacent to C or E. The low order bits of A, C and E are grouped, and the low order bits of B, D and F are grouped.

A(0-7) = C(0-7) = E(0-7)
B(0-7) = D(0-7) = F(0-7)

The high order bits of A and B are grouped A(8-17) = B(8-17), as are the high order bits of C and D as well as E and F. Lines of AB may occur in faces 0, 1, 4 and

5. In faces 4 and 5 the lines of AB occur in odd rows, while in faces 0 and 1 the lines are in even positions. The address bits 8–15 of AB may be computed from j, while bits 16 and 17 are computed from F. For odd line positions the address bits 8–15 are simply j shifted right one position. When the lines are in even positions the addresses are formed by adding one to j then shifting the result right by 1–bit. More precisely:

AB(8–15) = j>>1	if face = 4 or 5;
AB(8–15) = (j+1)>>1	if face = 0 or 1;

Similarly the faces of CD and EF are grouped.

CD(8–15) = j>>1	if face = 0 or 1;
CD(8–15) = (j+1)>>1	if face = 2 or 3;
EF(8–15) = j>>1	if face = 2 or 3;
EF(8–15) = (j+1)>>1	if face = 4 or 5.

As stated the two highest order bits of AB, CD and EF (bits 16 and 17) are computed from the face number directly.

The memory accessing function for retrieving memory values for adjacent pixels can be succinctly described in connection with a simplified example using only two adjacent pixels. Say the display memory is organised so that even location pixels in a raster scan line are stored in chip A and location pixels stored in chip B. Thus a row of pixels would have display memory addresses which alternative between chips A and B for adjacent pixels, ABABABABABABAB ..... etc. Assuming that the row is 512 pixels wide (i.e. 9–bits of address) means that there are 256 A pixels and 256 B pixels, hence the addressing of the A and B chips is only 8–bits. A pixel in B at a particular address is positioned to the right of the pixel in A at the same address.

So, if we know the location of where we want the adjacent pixels to 13–bits (from the vector intersection calculation) say 0011010110110, we truncate the top 8–bits to address B ie 00110101, and this is the location of one of the adjacent pixels. The other pixel is in A and is found by rounding to 8–bits, ie at location 001101110, (need to add 1 to the top 8–bits). The lower 4–bits then tell us the relative location between the two pixels and allows us to interpolate the pixel values as discussed above.

If we have had location 0011010100110 we would use the top 8-bits for A and B, without any additions.

The additions required to access adjacent pixels form yet another stage of the address recalculation pipeline and the actual adders are located on the display memory boards. The additions required for adjacent access and the address multiplexing required for double buffering are implemented with a set of four Erasable Programmable Logic Devices (EPLD) per render board. An overview of the EPLD is given in Figure 24. Figure 25 shows how the EPLDs are arranged to form two sets of addresses (for double buffering). This architecture allows six pixels to be addressed simultaneously where four of the six pixels are adjacent.

At the start of each display memory swap the renderer must clear all the RGB and Z components of each pixel in the display memory. As well as being able to read six pixels simultaneously the EPLD may write to six pixels simultaneously. When clearing the display memory contents after a double buffer swap the scan converter may write the same initialisation values to six pixels at one time. As a result the display memory clear time is reduced by a factor of six.

The display system 10′ enables the efficient implementation of virtual reality applications without the serious trade-off between latency and image resolution which has plagued most existing systems. What tended to suffer was frame rate, and hence latency, when the computer system could not keep up with the processing load. In such systems the frame rate would drop as the changes to the scene increased. More sophisticated systems would drop the accuracy of the virtual world model to minimise changes which require re-rendering. Other systems would reduce the accuracy of their rendering to try to maintain a minimum frame rate. All such systems degrade rapidly when stressed by change in the view of the virtual environment.

In the present system 10′ latency to rotation is independent of the virtual world;

it is a function of hardware and is effectively limited by the tracking equipment. Translational latency is reduced indirectly in that distant objects in the background will not be observed to move and need not be re-rendered. Changes in the environment itself must still be handled, but by grouping the changes according to their observed speeds it is possible to minimise the number of display surfaces which need to be updated. In the same way changes due to translations and due to preserving stereoscopy can be grouped according to the required update rates and allocated to the display memories appropriately.

Although embodiments of the present invention present a solution to the extremely important and difficult case of rotational latency, translations and actual virtual environmental changes can conceivably overload the system. The important thing to note is that such problems would result in out of date representations of less important parts of the virtual world as seen by the user. It would not affect the smooth motion by lowering the frame rate and would not let the image quality suffer. Such an overload condition can only occur during very rapid translation or when there are fast moving objects in the virtual environment. In such cases the ability of the human eye to perceive much detail is limited since fast moving objects tend to blur. Hence it is unlikely that such overload would even be noticed by the users unless the system was grossly under powered.

The address recalculation pipeline is a graphics controller which removes the latency to head rotations which plague most virtual reality implementations. When used in conjunction with image composition the address recalculation pipeline may also lead to significant savings in the rendering required to maintain the illusion of immersion within a virtual world. The techniques of image composition and prioritised rendering are most useful in the realm of virtual reality due to the independence of the rendered image and the user's head orientation when using the address recalculation pipeline. Prioritised rendering in particular enables reductions in rendering of orders of magnitude and also allows effective exploitation of parallel rendering hardware. The performance achievable by such a system is well beyond that attainable with conventional graphics computer systems, even those at the high-end of the market.

The cost of a virtual reality system with an address recalculation pipeline is actually considerably lower than trying to achieve equivalent performance with conventional graphics architectures. The address recalculation pipeline operates on the principle of having a display memory which surrounds the user rather than appearing as a rectangular screen in front of the user. Thus much more memory is required than for a conventional graphics system. As well there is a need for a number of large fast look-up tables which provide inputs at various stages of the pipeline. The access patterns for the display memory are rather random compared with the linear access patterns for conventional systems. This necessitates the use of large, fast, and relatively expensive static memories for the display memory. To allow for image composition and prioritised rendering multiple display memories are required, also providing potential parallelism. All in all over 100 Mbytes of static memory chips have been used in an implementation of the system 10′ which demonstrates performance which eludes commercial systems costing over a million dollars.

Another consequence of the address recalculation pipeline is the desirability for hardware anti-aliasing. Since the hardware employs a sampling technique image aliases are introduced. The nature of the processing means that software cannot perform the required anti-aliasing hence the hardware pipeline includes anti-aliasing stages. The anti-aliasing imposes further demands on the display memory systems, in particular requiring a number of adjacent pixels to be processed concurrently.

Prioritised rendering can use the expected validity time of the object's displayed image to determine an appropriate update rate. If the validity time is very long that implies that the object's displayed image is changing very slowly. Often that means that the object's change of appearance is not the focus of the user's attention and hence slight delays in updating that object will not be noticed. Of course in such cases there is not much rendering of the object going on anyway, but it may be useful in the extreme case of an overloaded system to know that very slowly changing things can be deferred in favour of more quickly changing objects which are more likely to be the focus of the user's attention.

At the other extreme, objects with very short expected validity times indicate very rapid changes in the object itself or its relative position with respect to the user. While such objects may indeed be the focus of attention, if they are changing or apparently moving at great speeds then we can rely on the human visual system not be able to perceive accurately the details of fast moving objects. Factors such as motion blur come into play. We can thus take advantage of the lack of accurate perception of fast changing images to allow a reduction of rendering in this case. Certainly in this case the rendering load of the object will be significant since the object has a high update rate. Human factors here should allow us to reduce the update rate if there is a shortage of rendering capacity without a significant perceived degradation in performance. It is certainly expected that one could reduce the fastest rate of update to about 25 to 30 Hz, motion picture and television update rates, if necessary, without user perceptible degradation. Of course update rates greater than the update rate of the display device cannot have any observable effects and should be avoided.

When the system is pushed to its limits in terms of computational capacity available for rendering objects, there is the possibility of taking advantage of the user's focus of attention to allow selective degradation of image resolution on peripheral or background objects. This is possible due to having multiple display memories with individual renderers which may not only operate at different update rates but in extreme situations may be operated at different resolutions.

One may also consider the physical nature of the user's body. Consider the human factors of head rotation. While very fast rotation about the neck is possible, the degree of rotation is limited. It is not possible to rotate your head completely. To look behind you requires a body rotation which is much slower than a head rotation. Also sudden reversal of rotation is limited by human physical factors, as is vertical rotation. It thus follows that the update rates for parts of the virtual world which cannot be viewed immediately can be reduced, hence reducing the rendering load. In other words the priority of carrying out those updates can be lower.

It is also relevant that humans cannot tolerate excessive acceleration. Vehicles too cannot change direction very rapidly, so it is possible to use low priority updates for objects behind and to the side of the user, concentrating on what is ahead in the direction of travel. Too much perceived acceleration in a virtual environment can induce the disconcerting effect known as vection.

Apart from merely travelling within the virtual world acting as an observer, users will want to manipulate objects within the virtual world and have other intentional interactions. Such manipulations may be achieved through gesture or more directly through handling of objects via flow-like devices. When performing such manipulation the user is generally concentrating on the task at hand and not moving much within the virtual world. It is therefore possible for the priority of updates of the objects being manipulated and the objects representing the manipulator to be increased significantly above those of the rest of the visible scene, ensuring prompt visual feedback. Because in such cases it is known what the focus of the user's attention is, in the extreme case of renderer overload it is possible to concentrate on the important aspects of the displayed scene.

These human factors can certainly be used to improve the effective performance of virtual reality implementations using the techniques described hereinabove.

With the improved performance available using the techniques described herein it is envisaged expected that the virtual worlds created by virtual reality applications will be characterised by complex scenes with highly realistic backgrounds and large numbers of independent objects. Conventional virtual reality systems suffer from unacceptable performance degradation with this kind of virtual world.

The technology also is directly applicable to telerobotic applications such as those found in controlling remote robotic manipulators in space or under sea. It is also ideal for Augmented Reality applications where virtual world images are superimposed on the real world.

U1-6	74ACT245	U80-85	74ACT374	U167	74F04
U7,U8	74F08	U86-97	74F283	U168,U169	74ACT374
U9	74F138	U98-103	74ACT374	U170,U171	74F32
U1O,U11	74ACT245	U104-115	74F283	U172	74ACT374
U12-23	MC51004P-45	U116-127	74ACT374	U173	74F00
U24-28	74ACT374	U128-130	CY7C516-40	U174	74F32
U29-40	MC51004P-45	U131	74ACT374		
U41-48	74ACT374	U132,U133	MC6206-20		
U49	CY7C516-40	U134	74ACT245		
U50,U51	74ACT374	U135	74ACT374		
U52	CY7C516-40	U136,U137	MC6206-20		
U53,U54	74ACT374	U138	74ACT245		
U55	CY7C516-40	U139	74ACT374		
U56	74F138	U140,U141	MC6206-20		
U58	74ACT245	U142	74ACT245		
U59,U60	74ACT374	U143-145	CY7C516-40		
U61	CY7C516-40	U146,U147	74ACT374		
U62,U63	74ACT374	U148	74ACT245		
U64	CY7C516-40	U149	74ACT374		
U65,U66	74ACT374	U150	74ACT245		
U67	CY7C516-40	U151	74ACT374		
U68	74F138	U152	74ACT245		
U69	74F08	U153,U154	74F86		
U70	74F138	U155	74F04		
U71,U72	74ACT374	U156-161	MC6206-20		
U73	CY7C516-40	U162	74F32		
U74,U75	74ACT374	U163	74F04		
U76	CY7C516-40	U164	74F32		
U77,U78	74ACT374	U165	74F04		
U79	CY7C516-40	U166	74F32		

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A method for selecting image pixel data for an image to be displayed comprising:

determining a rotational orientation for the image to be displayed; and

addressing stored image pixel data on the basis of the determined rotational orientation.

2. A method as claimed in claim 1, including the step of storing said image pixel data representing a panoramic scene mapped onto a notional surface encapsulating said fixed viewpoint.

3. A method as claimed in claim 2, wherein the step of determining the image orientation comprises sensing an orientation of an image viewing means.

4. A method as claimed in claim 3, wherein the image viewing means comprises a head–mounted display device for displaying the selected image pixel data.

5. A method as claimed in claim 4, including the steps of:

computing a first directional vector for the orientation of the head–mounted display device so as to determine an orientation for the image to be displayed;

determining an intersection point of said directional vector from said fixed viewpoint with said notional surface; and

mapping said intersection point onto said stored image pixel data so as to selectively address image pixel data for the image to be displayed.

6. A method as claimed in claim 5, wherein the selectively addressed image pixel data is displayed on said head–mounted display device.

7. A method as claimed in claim 6, including the steps of:

computing a second directional vector representing a pixel location of the head–mounted display device with respect to a user viewpoint thereof; and combining the first directional vector and the second directional vector to obtain a resultant vector, and determining an intersection point of said resultant vector from said fixed viewpoint with said notional surface.

8. A method as claimed in claim 7, including the step of applying a transformation to said second directional vector to compensate for a lens distortion of said head–mounted display device.

9. A method as claimed in any one of claims 2 to 8, wherein said notional surface is cuboid and substantially centred at said fixed viewpoint.

10. A method for displaying a pixel on a display means, comprising the steps of:

computing a first vector indicative of a pixel location on said display means with respect to a viewpoint for the display means;

computing a second vector indicative of a spatial orientation of the display means;

combining said first and second vectors into a resultant vector;

determining an intersection point of the resultant vector with a substantially closed notional surface when projected from a fixed interior projection point; and

mapping said intersection point onto stored image data so as to address image data for display at said pixel location.

11. A method as claimed in claim 10, wherein said notional surface is cuboid with said projection point at the centre thereof, and wherein said stored image data comprises image data representing a panoramic scene and mapped from a cuboid topology into addressable display memory.

12. A method as claimed in claim 10, wherein iterations of said steps are performed synchronously with the display of pixels on said display means.

13. A method as claimed in claim 12, wherein said steps are performed in a pipelined digital processing architecture which operates in synchronism with the display

of pixels on said display means.

14. A method as claimed in claim 10, including the step of transforming said first vector before combination with said second vector, to compensate for a lens distortion of the display means.

15. A method as claimed in any one of claims 11 to 14, wherein said panoramic scene is dynamic such that the stored image data includes first image data representing relatively slow moving objects and second image data representing relatively fast moving objects, the first and second image data being stored in separate display memories.

16. A method as claimed in claim 15, including the steps of:

periodically rendering and storing said first image data in said first display memory at a first update rate; and

periodically rendering and storing said second image data in said second display memory at a second update rate greater than said first update rate.

17. A method as claimed in claim 16, wherein said first and second image data includes a depth value, and wherein the method includes the step of selecting one of the first and second image data for display at a pixel location on said display means on the basis of a comparison as between the depth value for the first image data and the depth value for the second image data.

18. A method as claimed in claim 15, wherein image data representing said panoramic scene is stored in said first or second display memory on the basis of a calculated validity period representing a rate of change of size or position of the object.

19. A method as claimed in claim 18, wherein the validity period is calculated on the basis of a change of angle obtending across the object from the fixed projection point.

20. A method as claimed in claim 18, wherein the validity period is calculated on the basis of a change in angle obtending from the fixed projection point of the centre of the object.

21. A method as claimed in any one of claims 10 to 14, wherein the step of mapping said intersection point includes retrieving stored image data representing a plurality of adjacent pixels and interpolating the retrieved pixel values so as to generate a pixel value for display at said pixel location.

22. A method as claimed in claim 21, wherein the mapping of said intersection point utilises orthogonal position addresses each having a plurality of most significant bits and a plurality of least significant bits, the most significant bits of the position addresses being employed, in use, to address the stored image data adjacent pixels and the least significant bits being, in use, employed to interpolate the adjacent pixel image data to obtain a said pixel value for display at said pixel location.

23. A method as claimed in claim 21, wherein image data representing four adjacent pixels are retrieved simultaneously.

24. A method as claimed in claim 23, wherein the image data representing the four adjacent pixels are stored in respective individually addressable memories.

25. A method for displaying an image comprising:

storing image data representing images visible from at least substantially any rotational orientation of a fixed viewpoint;

determining a rotational orientation of a viewing means;

selecting a portion of said stored image data on the basis of the determined rotational orientation; and

displaying an image with said viewing means utilising said selected image data.

26. A method for displaying an image on a viewing means comprising:

storing image data representing images projected onto a surface at least substantially encapsulating a central viewpoint;

selecting a portion of said stored image data on the basis of an orientation of the viewing means; and

displaying an image with said viewing means using the selected image data portion.

27. A method for displaying an image to a user comprising:

storing image data representing views for at least substantially any rotational orientation of the user;

sensing a rotational orientation of the user;

selecting a portion of said image data on the basis of said sensed rotational orientation; and

displaying an image to the user utilising said selected image data.

28. A method for addressing image pixel data comprising:

generating and storing image pixel data representing a panoramic view encompassing views from a plurality of rotational orientations from a fixed viewpoint;

sensing a rotational orientation of a viewing means; and

addressing pixels of the stored image data on the basis of the sensed rotational orientation and raster scanning signals for the viewing means.

29. A method for reducing rotational latency in a head-mounted graphical display system, such as for virtual reality applications comprising:

generating and storing digitised image data for reproducing images representing a plurality of orientational views of a scene visible from a fixed viewpoint;

sensing the orientation of a head-mounted display means; and

selecting image pixel data from the digitised image data for display on the head-mounted display means, the image pixel data being selected synchronously with the display thereof and on the basis of said sensed orientation.

30. A method for generating pixel control signals for a graphical display means comprising:

generating pixel vector data on the basis of a pixel screen location of the display means and rotational orientation data;

addressing stored image data corresponding to said pixel screen location and said rotational orientation data on the basis of said pixel vector data; and

accessing the addressed image data and generating pixel control signals therefrom.

31. An apparatus for displaying a digitised graphical image comprising:

a storage means for storing image pixel data representing a panoramic view from a viewpoint;

a head-mounted display means including a display device and means for generating an orientation signal representative of the rotational orientation of the display means; and

means for selecting image pixel data from the storage means on the basis of said orientation signal and a pixel scanning signal for the display means, and generating display signals for controlling said display device on the basis of the selected image pixel data.

32. An apparatus as claimed in claim 31, wherein said storage means comprises first and second memories for storing respective image pixel data representing panoramic views from spatially displaced said viewpoints.

33. An apparatus as claimed in claim 31, wherein the panoramic view comprises a plurality of adjoining orthogonal views from said fixed viewpoint, the stored image pixel data for the adjoining orthogonal views being mappable onto a cuboid surface.

34. An apparatus as claimed in claim 33, wherein the storage means and the means for selecting image pixel data comprise a pipelined digital processing architecture.

35. An apparatus as claimed in claim 33, wherein the display signal is generated by the pipeline synchronously with the pixel scanning signal for the display means.

36. An apparatus as claimed in claim 34, wherein said pipeline comprises a first stage for generating first vector data representing a vector in the direction of a pixel indicated by the pixel scanning signal from a viewpoint for the display device.

37. An apparatus as claimed in claim 36, wherein the pipeline first stage includes means for compensating said first vector data for a lens distortion of the head-mounted display means.

38. An apparatus as claimed in claim 36, wherein the pipeline first stage comprises a look-up table which utilises the pixel scanning signal representing a pixel display location as input and produces said first vector data as output.

39. An apparatus as claimed in claim 38, wherein the pipeline comprises a second stage for generating resultant vector data from said first vector data and said orientation signal from the display means.

40. An apparatus as claimed in claim 39, wherein said orientation signal conveys orthogonal vector data indicative of the rotational orientation of the display means, the pipeline second stage performing a matrix multiplication operation to generate said resultant vector data from the first vector data and the display means orthogonal vector data.

41. An apparatus as claimed in claim 39, wherein the pipeline comprises a third stage for generating an address on the basis of said resultant vector data for selecting image pixel data from the storage means.

42. An apparatus as claimed in claim 41, wherein the pipeline third stage comprises intersection means for determining an intersection point of the vector represented by the resultant vector data with a said cuboid surface when projected from a fixed interior projection point.

43. An apparatus as claimed in claim 42, wherein the pipeline third stage produces

at least one address corresponding to image pixel data adjacent said intersection point when mapped onto said cuboid surface.

44. An apparatus as claimed in claim 43, wherein the pipeline includes a fourth stage comprising said storage means which outputs selected image pixel data on the basis of the at least one address from the pipeline third stage.

45. An apparatus as claimed in claim 44, wherein the pipeline third stage in use generates a plurality of addresses corresponding to the four closest pixels to the intersection point of the image pixel data when mapped onto said cuboid surface, and wherein the pipeline comprises a fifth stage for interpolating the four closest image pixel data with respect to the intersection point.

46. An apparatus as claimed in claim 45, wherein the calculated intersection point comprises two orthogonal coordinate values each consisting of a plurality of most significant bits and a plurality of least significant bits, and wherein the pipeline third stage generates the plurality of addresses utilising the most significant bits of the coordinate values and the pipeline fifth stage interpolates the image pixel data retrieved from the storage means utilising the least significant bits of the coordinate values.

47. An apparatus as claimed in claim 45, wherein the storage means comprises at least four individually and synchronously addressable memory means for simultaneously accessing the four closest image pixel data values.

48. An apparatus as claimed in claim 47, wherein the storage means comprises six individually and simultaneously accessible memories such that each four image pixel data values which are adjacent when mapped onto said cuboid surface are simultaneously accessible.

49. An apparatus as claimed in any one of claims 31 to 48, wherein the panoramic view represented by the stored image pixel data is dynamic and wherein said storage means comprises at least one first memory for storing first image data representing

relatively slow moving objects and at least one second memory for storing second image data representing relatively fast moving objects.

50. An apparatus as claimed in claim 49, wherein the first and second image data includes a depth value, the apparatus including means for selecting one of the first and second image data for display at a pixel location on said display device on the basis of a comparison as between the depth value for the first image data and the depth value for the second image data.

51. An apparatus as claimed in claim 49, including at least one image rendering means for generating the first and second image data for storage in the respective first and second memories.

52. An apparatus as claimed in claim 51, including a means for calculating a validity period representing a rage of change of size or position of a said object for determining whether to store the image pixel data representing the object in said first or second memories.

53. An apparatus as claimed in claim 52, wherein the validity period is calculated on the basis of a change of angle obtending across the object from the fixed projection point.

54. An apparatus as claimed in claim 52, wherein the validity period is calculated on the basis of a change in angle obtending from the fixed projection point of the centre of the object.

55. A graphical simulation system comprising:

a digital memory for storing data representing a panoramic view rendered onto a notational surface at least substantially surrounding a viewpoint;

a display means capable of rotational orientation changes and including means for generating data indicative of the rotational orientation of the display means; and

a display controller for retrieving image data from said memory in accordance with rotational orientation data from the display means so as to control the display means to generate an image representing a view from the viewpoint at the rotational orientation of the display means.

56. A display controlling apparatus for use with a display apparatus including imaging means for displaying pixilated image data to the eyes of a user and rotational orientation sensing means for sensing the rotational orientation of the user's head comprising:

a display memory buffer for storing image data representing a panoramic image rendered onto a notional surface substantially surrounding a viewpoint; and

a display controller for accessing image data in the display memory buffer in accordance with the sensed rotational orientation of the user's head, and for generating display signals for said imaging means to display an image corresponding to a view of said panoramic image from the sensed rotational orientation at the viewpoint.

57. A graphical display controller pipeline for generating pixel control signals for a display means comprising:

a first stage for generating pixel vector data on the basis of a pixel screen location of the display means and rotational orientation data;

a memory location conversion stage for addressing image data corresponding to said pixel screen location and rotational orientation data on the basis of said pixel vector data; and

a pixel generation stage for retrieving the addressed image data and generating therefrom pixel control signals for the display means.

58. A graphical display controller pipeline for generating pixel control signals for a head-mounted display means which includes a display device and rotational orientation sensing means comprising:

a first stage for generating pixel vector data on the basis of a pixel screen location for the display device;

a second stage for generating resultant vector data from the pixel vector data and the sensed rotational orientation of the display means;

a third stage for generating at least one display memory address from the resultant vector data;

a fourth stage for retrieving display pixel data from at least one display memory on the basis of the at least one display memory address; and

a fifth stage for generating pixel control signals for the display device on the basis of said display pixel data.

59. A graphical display controller pipeline as claimed in claim 58, wherein the pipeline third stage comprises intersection processing means for determining an intersection point of the vector represented by said resultant vector data with a notional cuboid surface when projected from a central interior viewpoint thereof, and means for mapping the intersection point onto said at least one display memory address.

60. A graphical display controller pipeline as claimed in claim 59, wherein the intersection point determined by the intersection processing means comprises a digital quantity, the most significant bits of which are mapped so as to address logically adjacent display pixel data values, the logically adjacent display pixel data values being, in use, retrieved simultaneously by the pipeline fourth stage, the least significant bits of the digital quantity being utilised by the pipeline fifth stage to interpolate the logically adjacent pixel data values and generate control signals therefrom for display of a single pixel on the display device.

61. A method for composing images in a graphic display system comprising:

periodically rendering and storing first image data in a first display memory at a first update rate;

periodically rendering and storing second image data in a second display memory at a second update rate; and

generating output image data on the basis of the first and second image data, wherein said first update rate is greater than said second update rate.

62. A method as claimed in claim 61, wherein the first and second image data include a depth field value, the step of generating output image data including selecting

one of the first and second image data on the basis of a comparison as between the depth field valves for the first and second image data.

63. A method as claimed in claim 61, including assigning image data for an object to one of the display memories on the basis of a list of objects ordered according to predetermined times the respective image data thereof is anticipated to remain valid.

64. A method as claimed in claim 63, wherein said list excludes objects allocated to a display memory with a low update rate.

65. A method as claimed in claim 63, wherein said assigning step includes selecting said display memories sequentially from the display memory with the fastest update rate to the display memory with the slowest update rate.

66. A method as claimed in claim 65, wherein a next display memory in the sequence is selected when a render performance threshold will be exceeded if a display memory is allocated said image data for said object.

67. A graphic display system comprising:

at least one graphic rendering means for generating image data corresponding to stored object display information for objects to be displayed by the system;

a plurality of display memories for storing image data generated by the at least one rendering means; and

selection means for determining which of said plurality of display memories to store image data for a particular object on the basis of a characteristic of that object.

68. A graphic display system as claimed in claim 61, wherein each of the plurality of display memories is periodically updated with image data from the rendering means, the length of period between updates being different as between different display memories.

69. A graphic display system as claimed in claim 68, including a plurality of display

memories are provided each having a different update period of image data from the rendering means, wherein the selection means determines the display memory to store image data for an object based on a computed validity period for the object.

70. A graphic display system as claimed in claim 69, wherein the computed validity period for an object is dependent upon relative movement of the object within the graphic scene which is to be displayed by the system.

71. A graphic display system as claimed in claim 69, wherein the selection means selects a display memory for an object which has an update period which is less than the validity period for the object.

72. A method as claimed in claim 68, wherein said selection means assigns image data for an object to one of the display memories on the basis of a list of objects ordered according to predetermined times the respective image data thereof is anticipated to remain valid.

73. A method as claimed in claim 72, wherein said list excludes objects allocated to a display memory with a low update rate.

74. A method as claimed in claim 72, wherein said selection means selects said display memories sequentially from the display memory with the fastest update rate to the display memory with the slowest update rate.

75. A method as claimed in claim 74, wherein a next display memory in the sequence is selected when a render performance threshold will be exceeded if a display memory is allocated said image data for said object.

76. An image data processing apparatus comprising:

a rendering engine for generating first and second image data;

first and second storage means for storing image data;

selection means for selecting which of the first and second image data to store in the respective first and second storage means; and

an image composition means for comparing equivalent pixels from said first and second image data and selecting therebetween for generation of output image data,

wherein said rendering engine generates the first and second image data at different average rates.

77. An image data processing system as claimed in claim 76, wherein the rendering engine generates the first and second image data periodically, the period being different as between the first and second image data.

78. A method as claimed in claim 77, wherein said selection means assigns image data for an object to one of the display memories on the basis of a list of objects ordered according to predetermined times the respective image data thereof is anticipated to remain valid.

79. A method as claimed in claim 78, wherein said list excludes objects allocated to a display memory with a low update rate.

80. A method as claimed in claim 78, wherein said selection means selects said display memories sequentially from the display memory with the fastest update rate to the display memory with the slowest update rate.

81. A method as claimed in claim 80, wherein a next display memory in the sequence is selected when a render performance threshold will be exceeded if a display memory is allocated said image data for said object.

82. An image data processing apparatus for generating output image data representing objects comprising a visual display, the apparatus comprising:

at least one rendering engine for generating image data representing objects comprising a visual display;

a plurality of display memories for storing image data generated by the at least one rendering engine, the display memories being, in use, periodically updated with image data with the update period being different as between different display memories;

a selection means for determining which of the plurality of display memories to store image data representing a particular object on the basis of a calculated validity period for that object indicative of relative movement of the object on the visual display; and

image composition means for retrieving image data from the plurality of display memories and generating output image data therefrom according to a comparison of depth field data for the retrieved image data indicative of the relative depths of the objects represented thereby within the visual display.

83. A method as claimed in claim 82, wherein said selection means assigns image data for an object to one of the display memories on the basis of a list of objects ordered according to predetermined times the respective image data thereof is anticipated to remain valid.

84. A method as claimed in claim 83, wherein said list excludes objects allocated to a display memory with a low update rate.

85. A method as claimed in claim 83, wherein said selection means selects said display memories sequentially from the display memory with the fastest update rate to the display memory with the slowest update rate.

86. A method as claimed in claim 85, wherein a next display memory in the sequence is selected when a render performance threshold will be exceeded if a display memory is allocated said image data for said object.

87. A method for processing stored image pixel data for display, the stored image pixel data being mappable onto a notional display surface, comprising the steps of:

calculating an intersection point of a viewport vector with said notional display surface;

accessing stored data for a plurality of pixels identified as adjacent the intersection point when mapped to said display surface; and

interpolating the accessed data on the basis of the relative positioning of said intersection point and the adjacent pixel mapping locations to generate an output pixel value.

88. A method as claimed in claim 87, wherein the most significant bits of the calculated intersection point are utilised for identifying the adjacent pixels and the least significant bits are utilised for interpolating the accessed data.

89. A method as claimed in claim 87 or 88, wherein the stored data for each of the respective pixels of said plurality of adjacent pixels are, stored in separate memory devices so as to be simultaneously accessible.

90. A display memory architecture wherein adjacent pixel values of a graphic image are stored in separate memory devices so as to be simultaneously accessible.

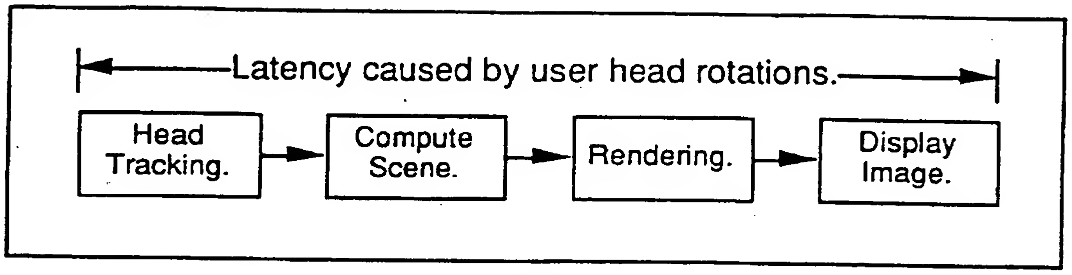


FIGURE 1

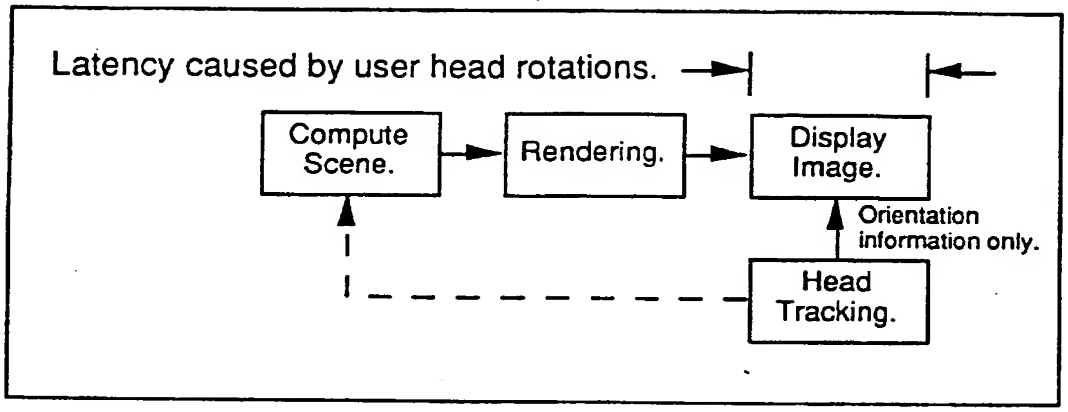

Latency caused by user head rotations.
Compute Scene.
Rendering.
Display Image.
Orientation information only.
Head Tracking.

FIGURE 2

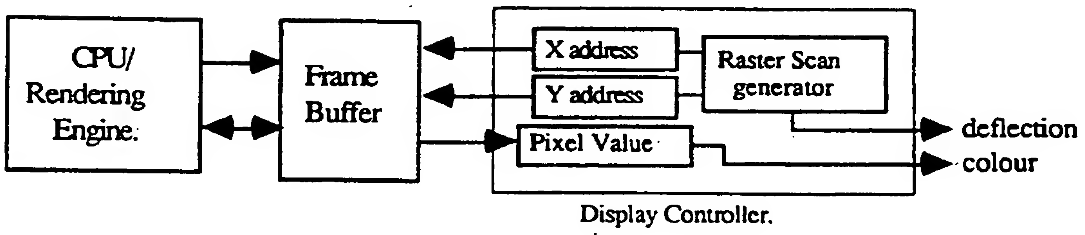


FIGURE 3

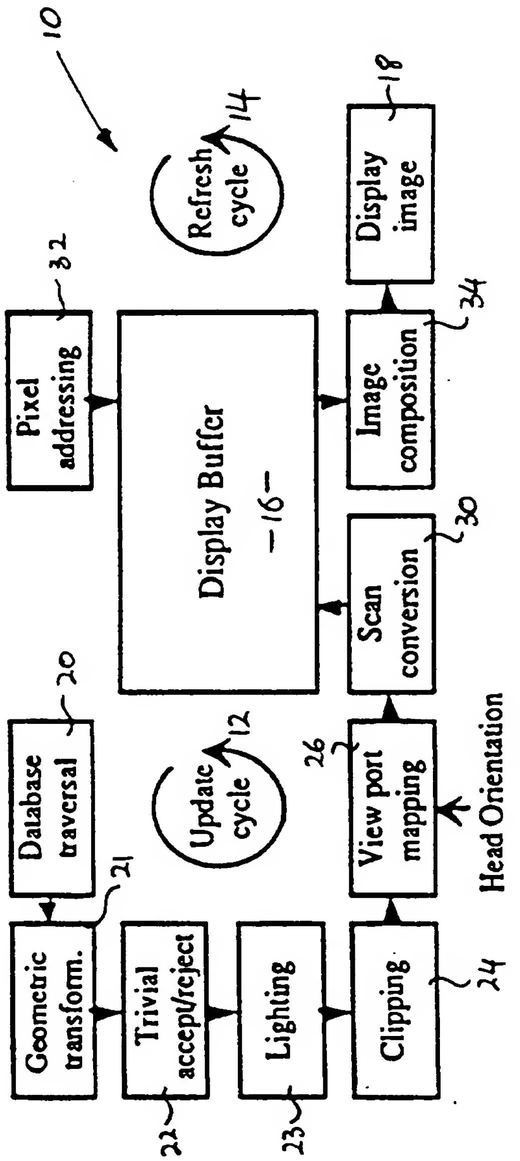


FIGURE 4

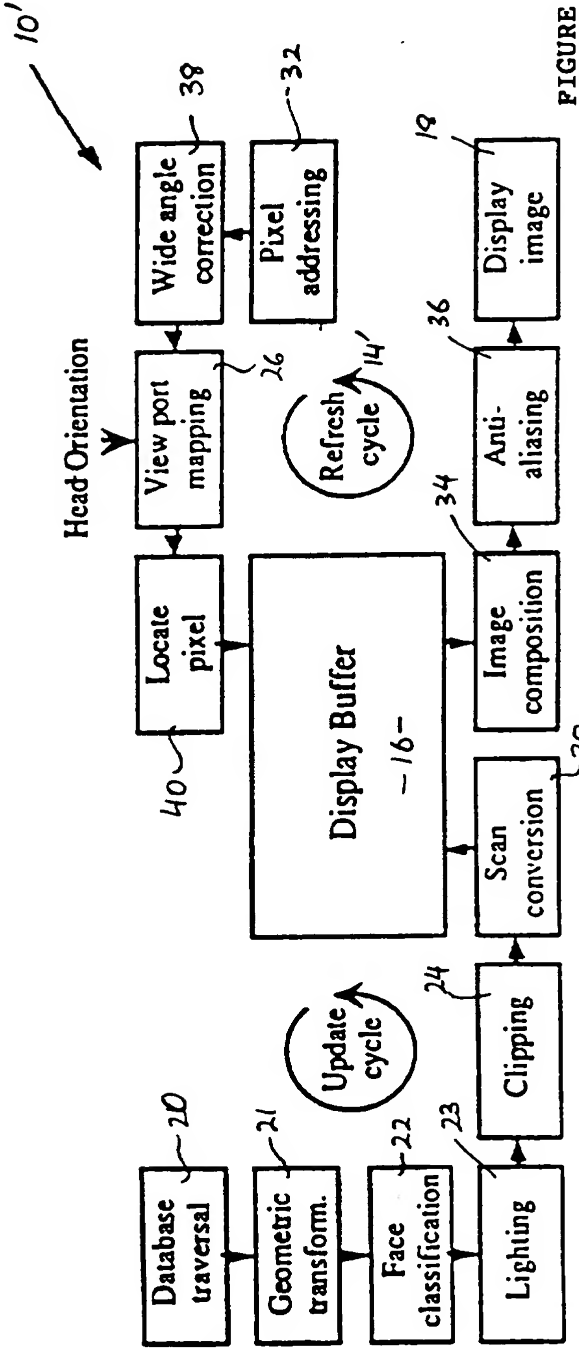
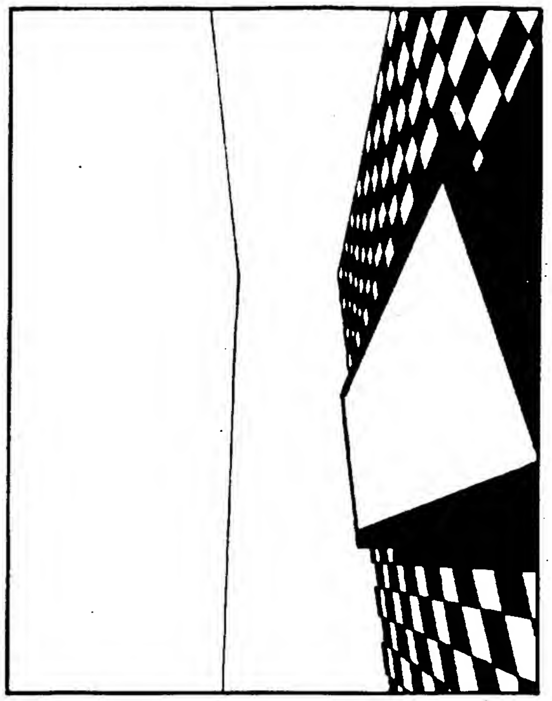


FIGURE 5



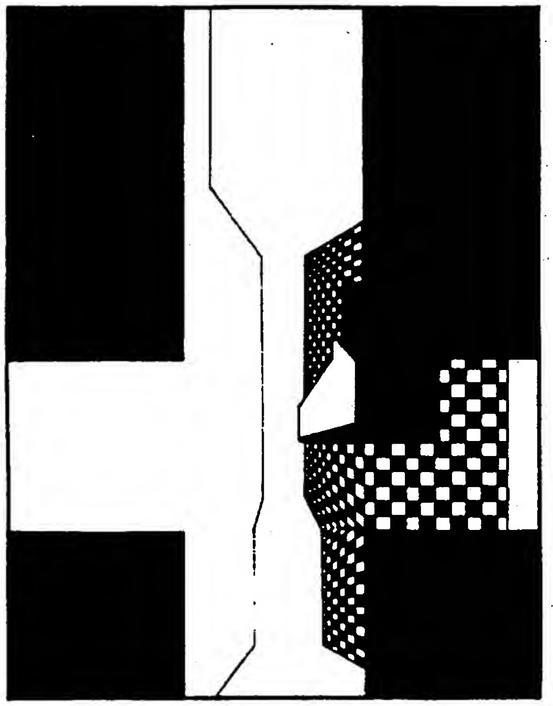


FIGURE 6

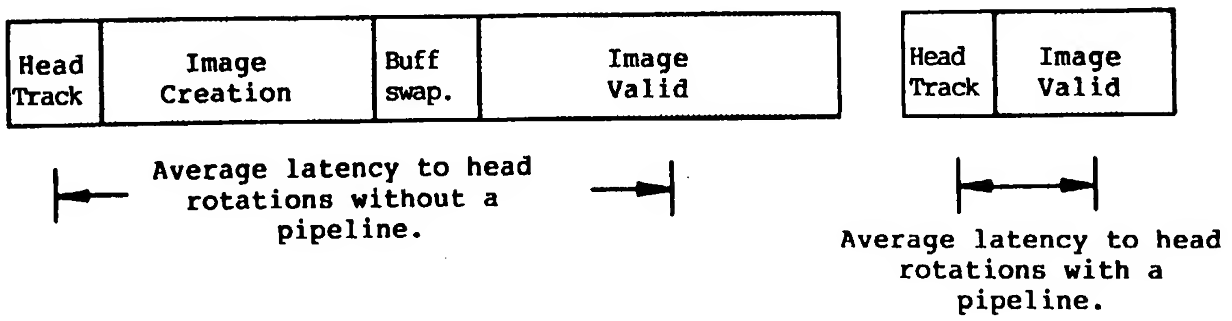

Head Track
Image Creation
Buff swap.
Image Valid
Average latency to head rotations without a pipeline.
Head Track
Image Valid
Average latency to head rotations with a pipeline.

FIGURE 7

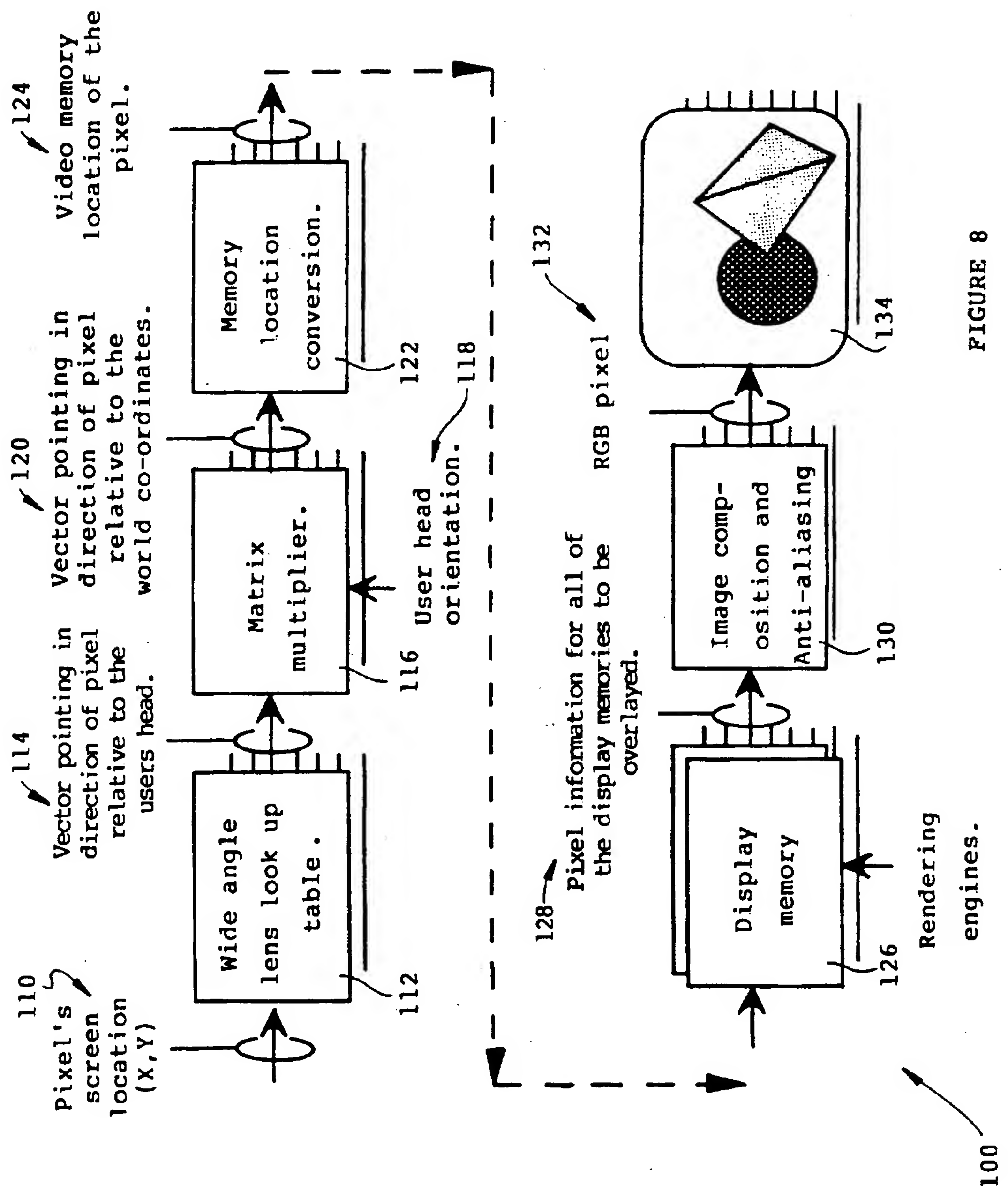

Pixel's screen location (X,Y)
110
Wide angle lens look up table.
112
114
Vector pointing in direction of pixel relative to the users head.
Matrix multiplier.
116
120
Vector pointing in direction of pixel relative to the world co-ordinates.
User head orientation.
118
Memory location conversion.
122
124
Video memory location of the pixel.
128
Pixel information for all of the display memories to be overlayed.
Display memory
126
Rendering engines.
Image composition and Anti-aliasing
130
RGB pixel
132
134
100

FIGURE 8

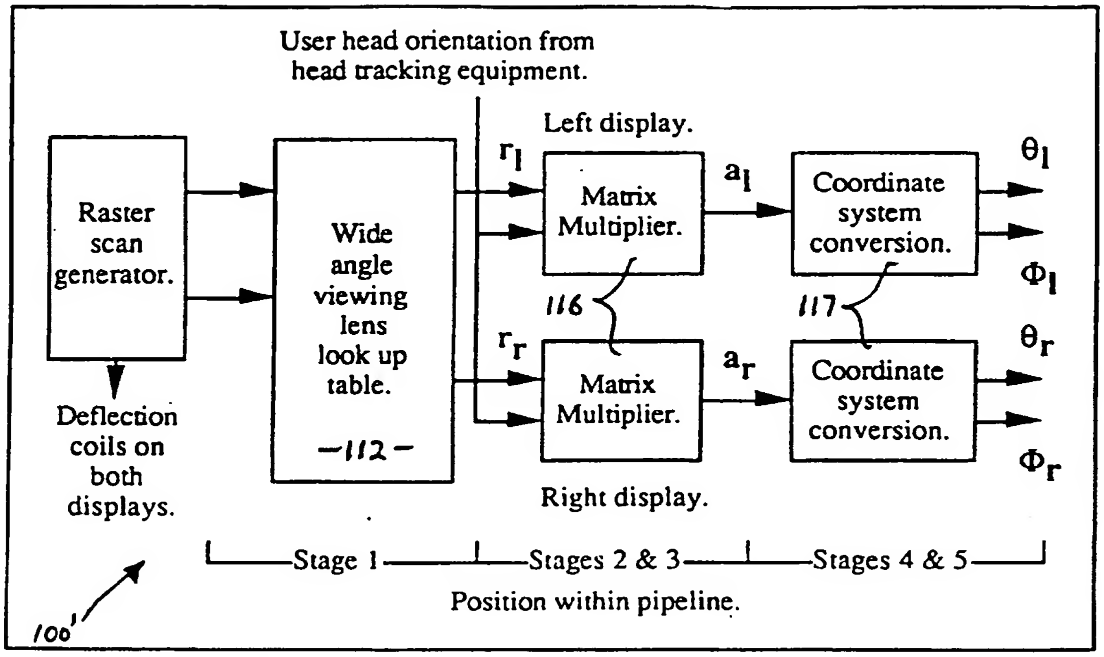

User head orientation from head tracking equipment.
Left display.
Raster scan generator.
Wide angle viewing lens look up table.
-112-
r_l
r_r
Matrix Multiplier.
Matrix Multiplier.
a_l
a_r
Coordinate system conversion.
Coordinate system conversion.
θ_l
Φ_l
θ_r
Φ_r
116
117
Deflection coils on both displays.
Right display.
Stage 1
Stages 2 & 3
Stages 4 & 5
Position within pipeline.
100'

FIGURE 9

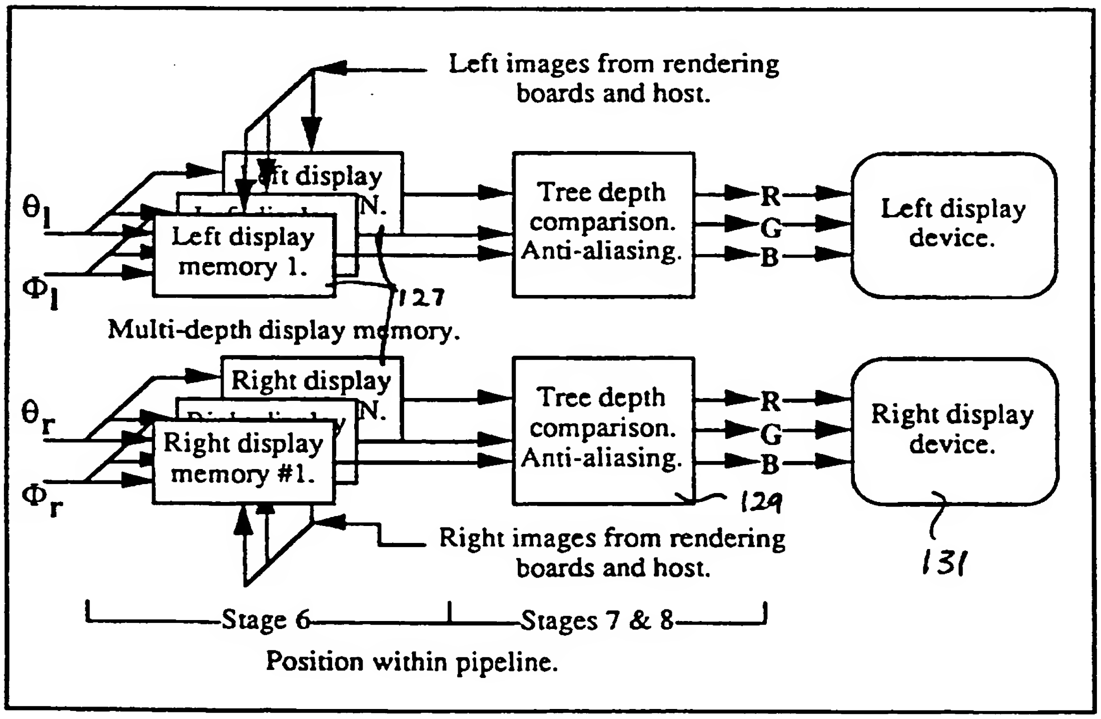

Left images from rendering boards and host.
θ_l
Φ_l
Left display ... N.
Left display memory 1.
Tree depth comparison. Anti-aliasing.
R
G
B
Left display device.
127
Multi-depth display memory.
θ_r
Φ_r
Right display ... N.
Right display memory #1.
Tree depth comparison. Anti-aliasing.
R
G
B
Right display device.
129
131
Right images from rendering boards and host.
Stage 6
Stages 7 & 8
Position within pipeline.

FIGURE 10

$$\begin{bmatrix} X_x & X_y & X_z \\ Y_x & Y_y & Y_z \\ Z_x & Z_y & Z_z \end{bmatrix} \bullet \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix}$$

150

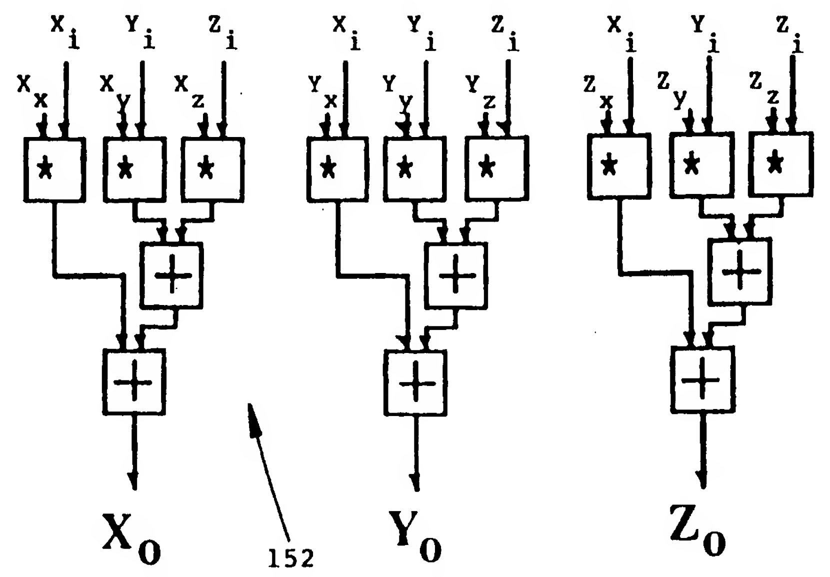


FIGURE 11

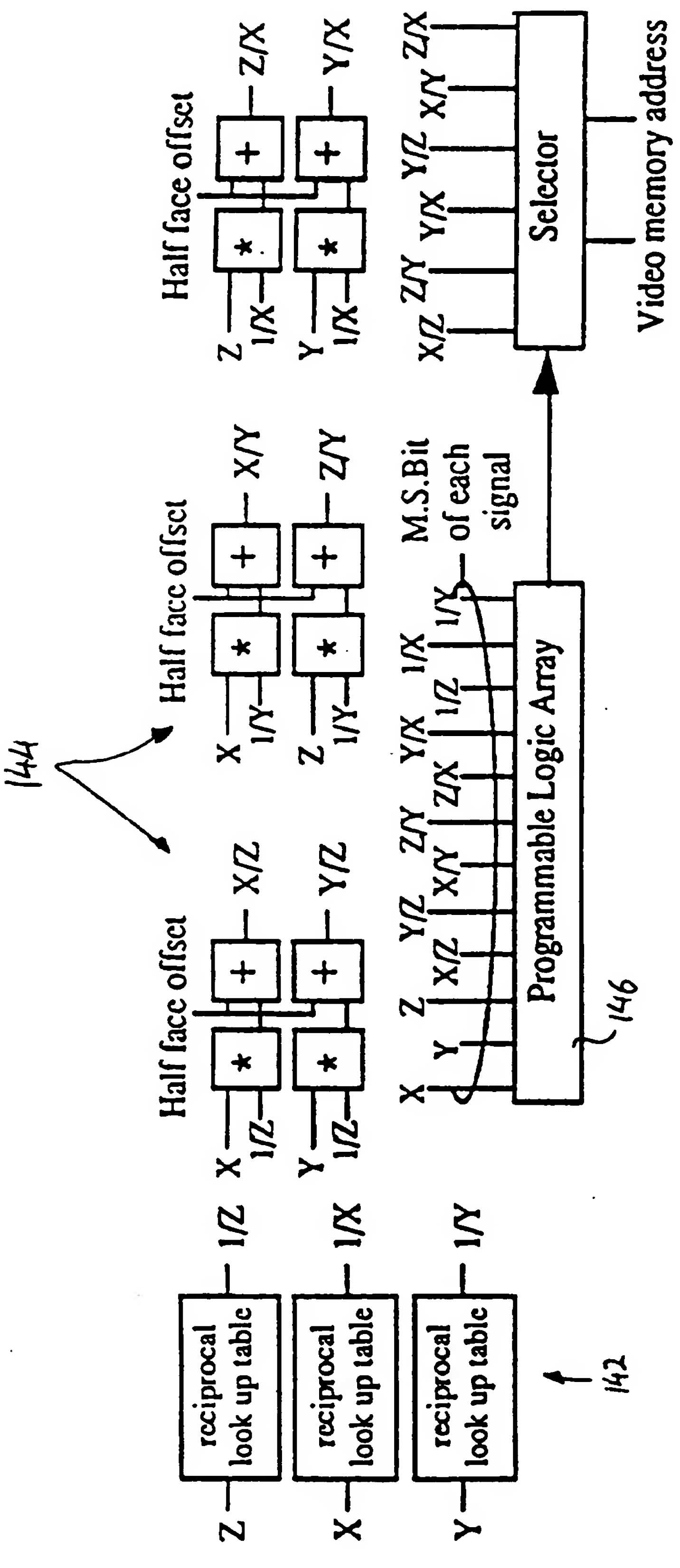

Z
reciprocal look up table
1/Z
X
reciprocal look up table
1/X
Y
reciprocal look up table
1/Y
142
144
Half face offset
X
1/Z
*
+
X/Z
Y
1/Z
*
+
Y/Z
Half face offset
X
1/Y
*
+
X/Y
Z
1/Y
*
+
Z/Y
Half face offset
Z
1/X
*
+
Z/X
Y
1/X
*
+
Y/X
X
Y
Z
X/Z
Y/Z
X/Y
Z/Y
Z/X
Y/X
1/Z
1/X
1/Y
M.S.Bit of each signal
Programmable Logic Array
146
X/Z
Z/Y
Y/X
Y/Z
X/Y
Z/X
Selector
Video memory address

FIGURE 12

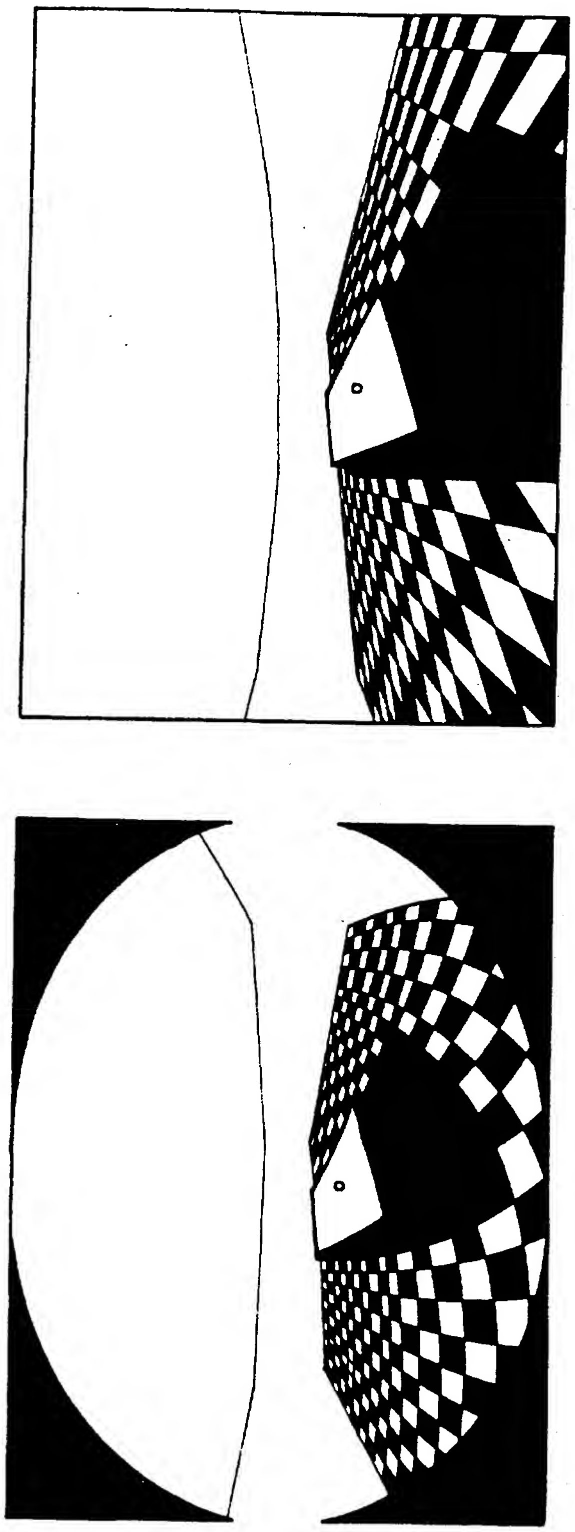


FIGURE 13

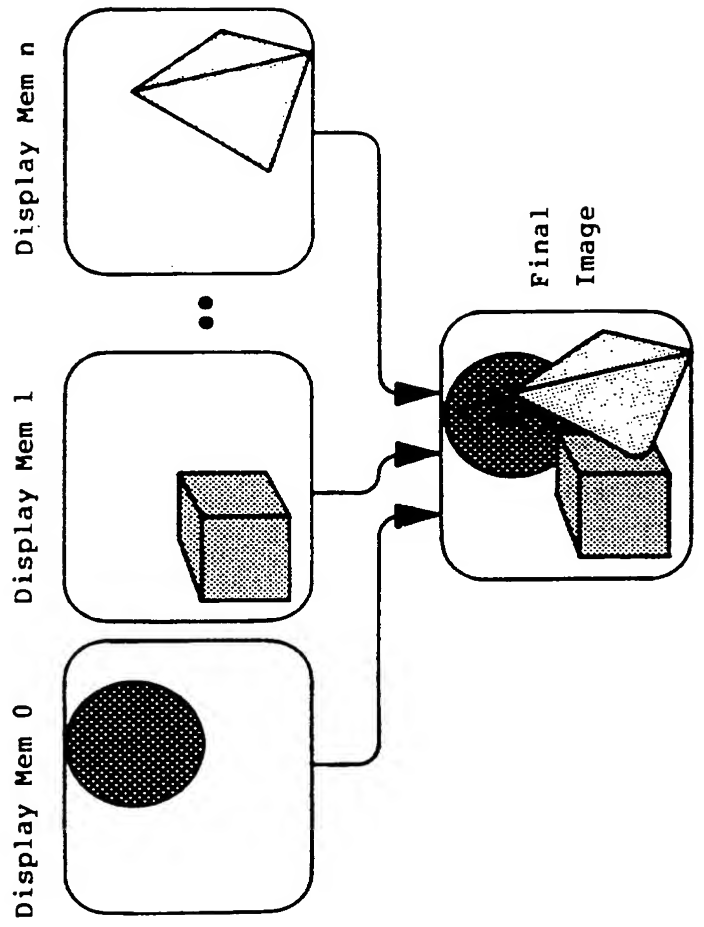
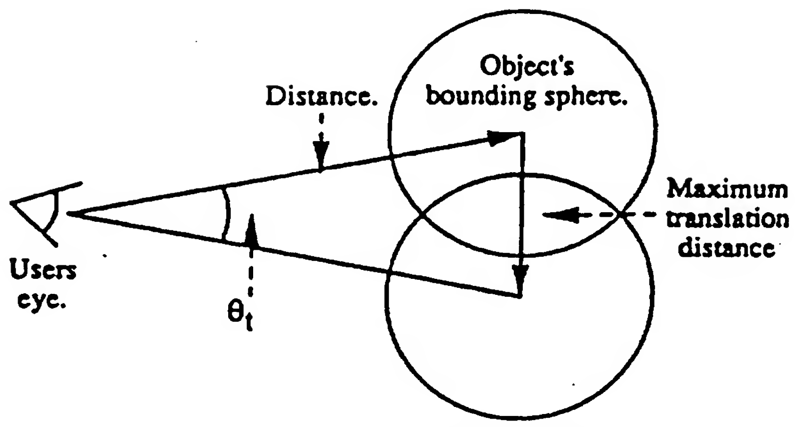


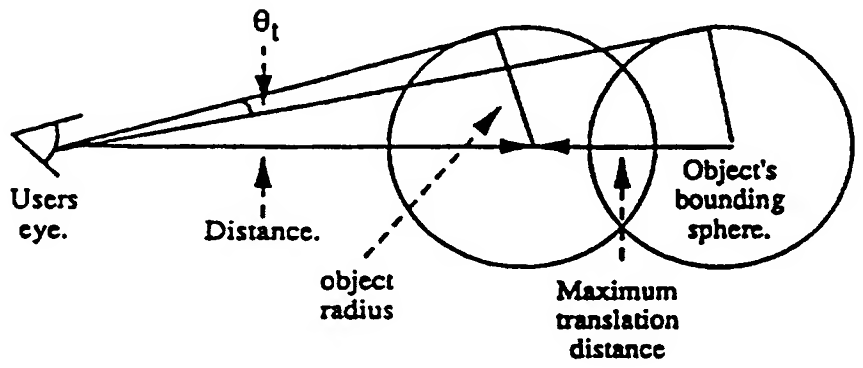
FIGURE 14



$$\tau_{translation}(i) = \frac{distance(i) * sqrt(2*(1-\cos(\theta_t)))}{relative\_speed(i)}$$

Where i = object number.

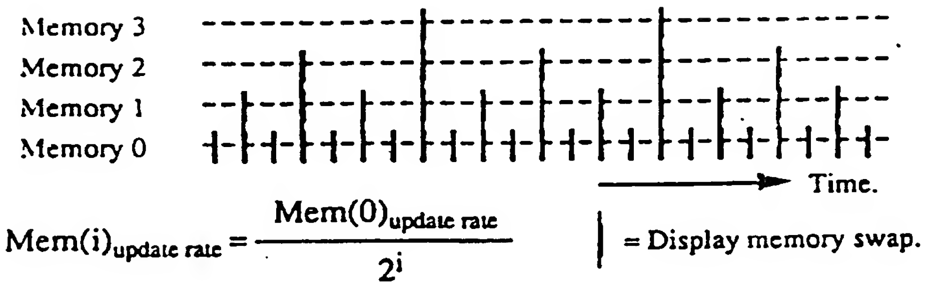
FIGURE 15



$$\tau_{size}(i) = \frac{distance(i) - radius(i)/\sin(\theta_t + Asin(radius(i)/distance(i)))}{relative\_speed(i)}$$

Where i = object number.

FIGURE 16



$$\mathrm{Mem}(i)_{\text{update rate}} = \frac{\mathrm{Mem}(0)_{\text{update rate}}}{2^i}$$

FIGURE 17

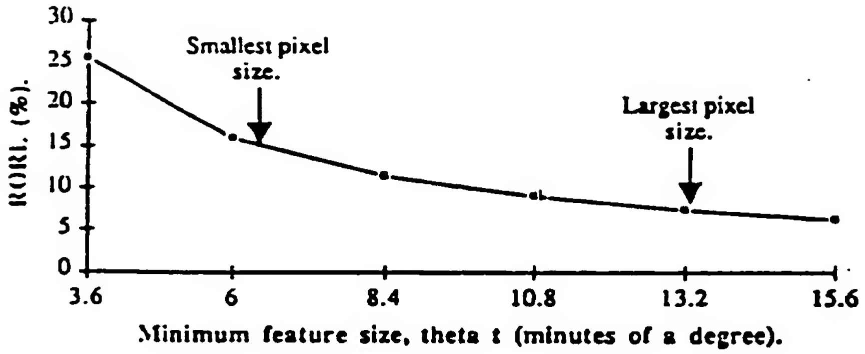


FIGURE 18

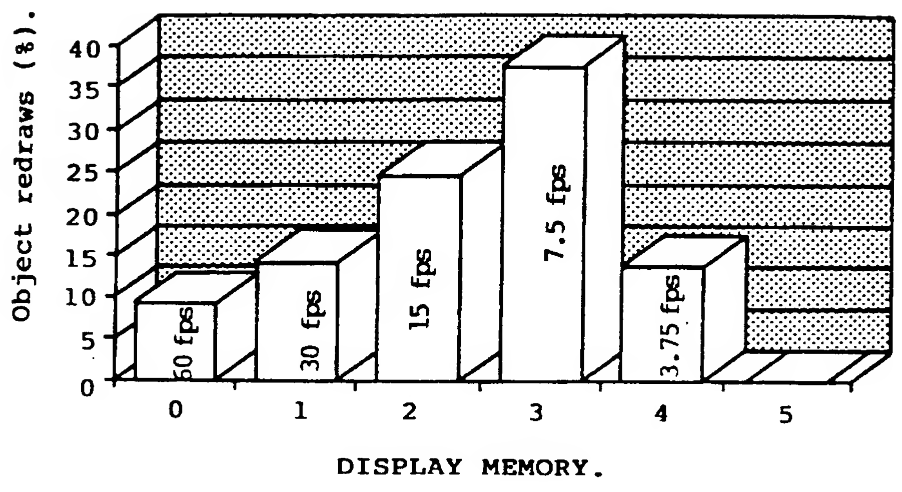


FIGURE 19

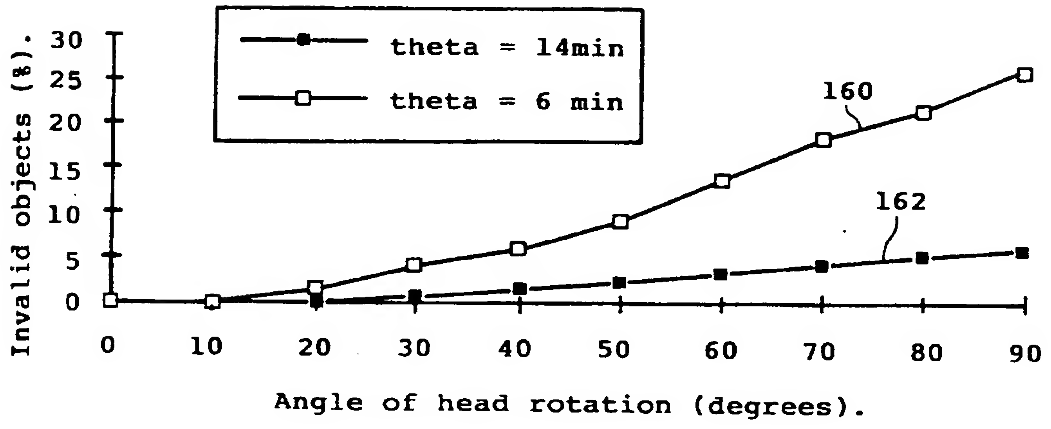


FIGURE 20

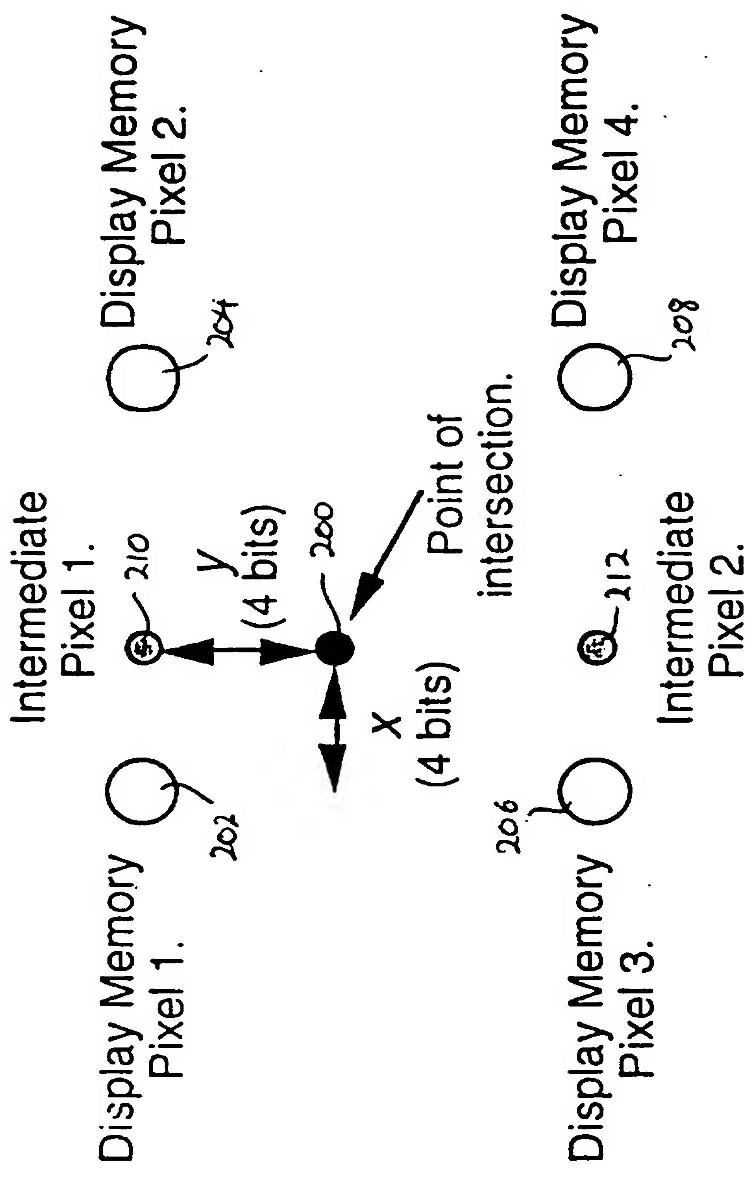

Display Memory Pixel 1.
202
Intermediate Pixel 1.
210
Display Memory Pixel 2.
204
x (4 bits)
y (4 bits)
200
Point of intersection.
Display Memory Pixel 3.
206
212
Intermediate Pixel 2.
Display Memory Pixel 4.
208

FIGURE 21

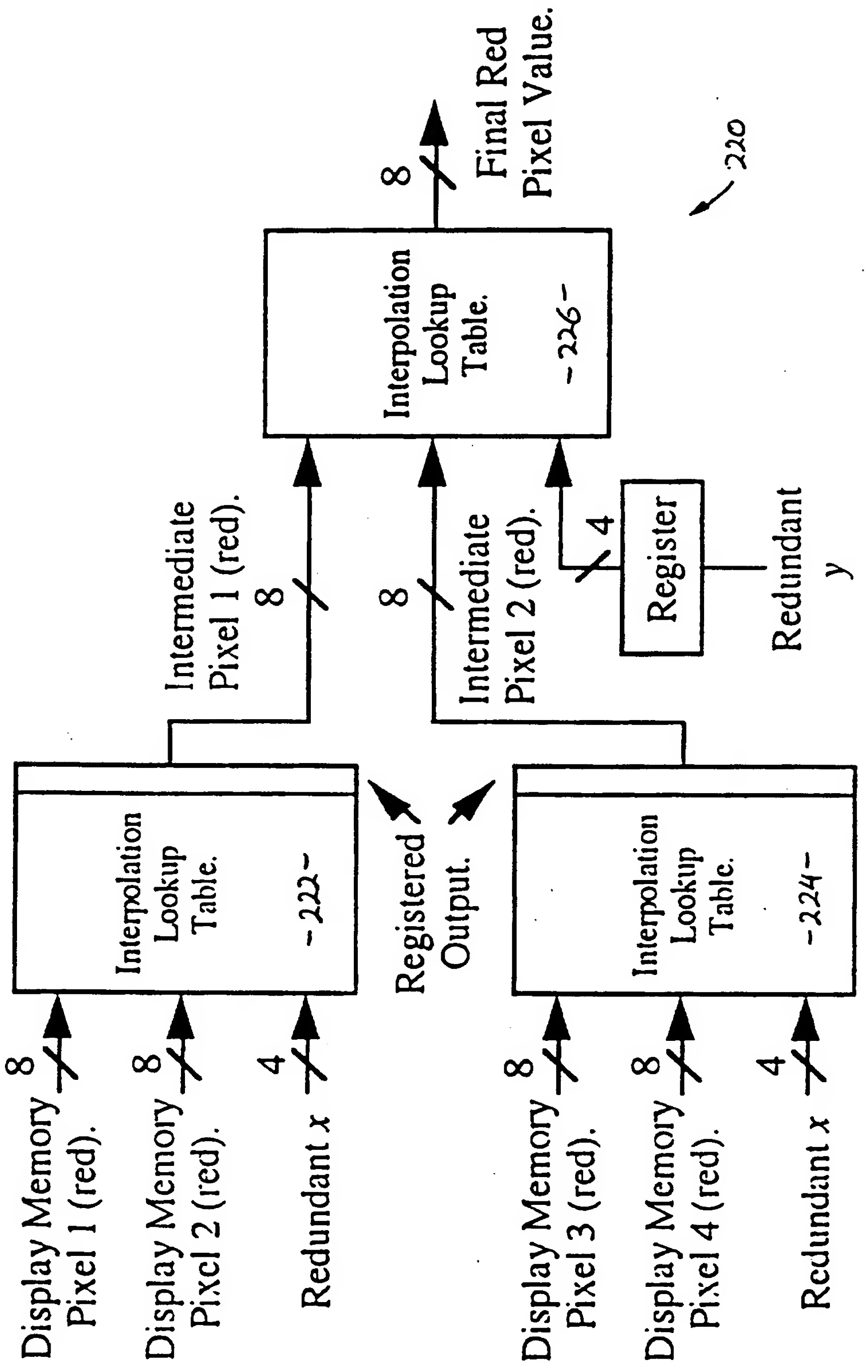

Display Memory Pixel 1 (red).
8
Display Memory Pixel 2 (red).
8
Redundant x
4
Interpolation Lookup Table.
-222-
Display Memory Pixel 3 (red).
8
Display Memory Pixel 4 (red).
8
Redundant x
4
Interpolation Lookup Table.
-224-
Registered Output.
Intermediate Pixel 1 (red).
8
Intermediate Pixel 2 (red).
8
Register
4
Redundant y
Interpolation Lookup Table.
-226-
8
Final Red Pixel Value.
220

FIGURE 22

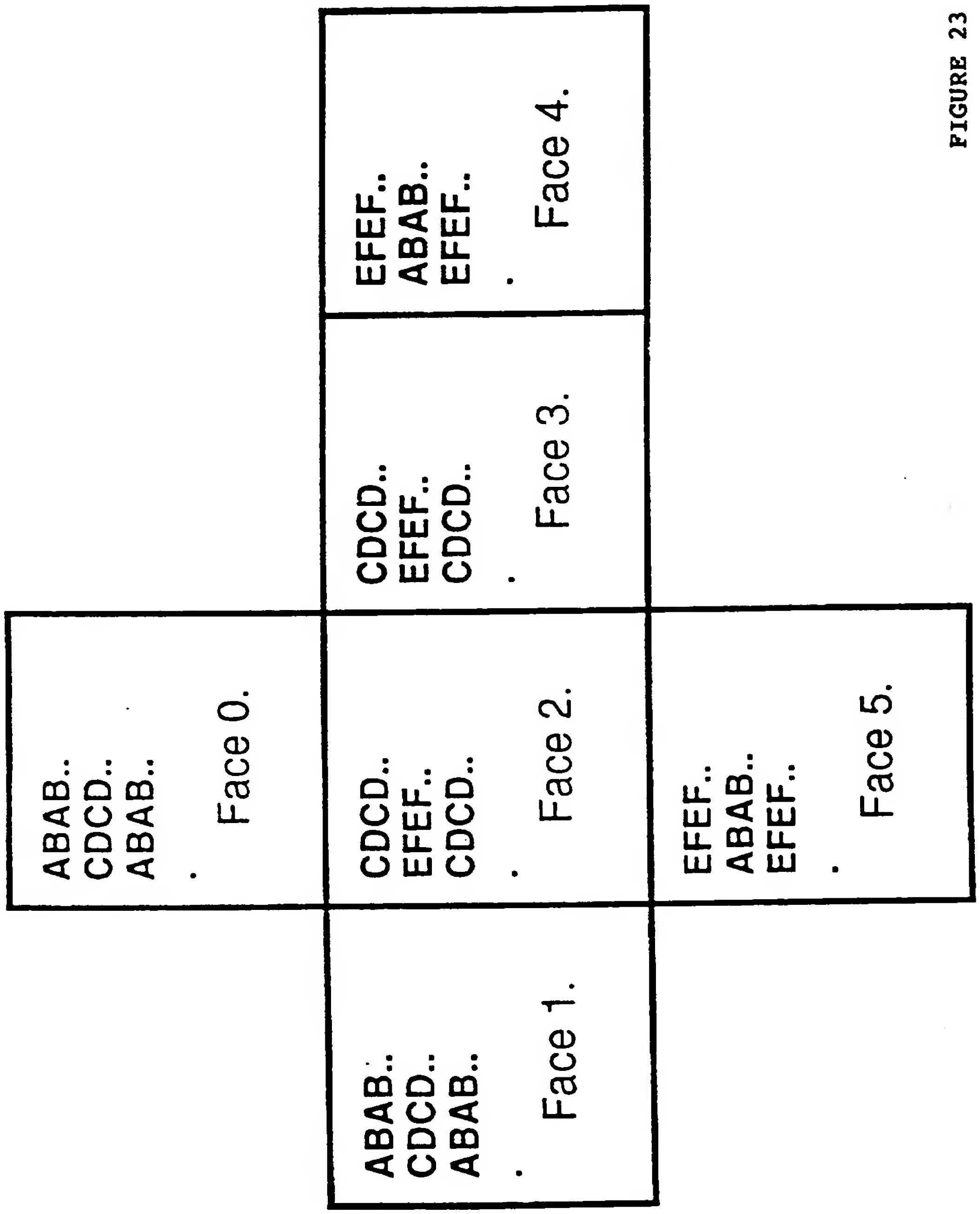

ABAB..
CDCD..
ABAB..
.
Face 0.
ABAB..
CDCD..
ABAB..
.
Face 1.
CDCD..
EFEF..
CDCD..
.
Face 2.
CDCD..
EFEF..
CDCD..
.
Face 3.
EFEF..
ABAB..
EFEF..
.
Face 4.
EFEF..
ABAB..
EFEF..
.
Face 5.

FIGURE 23

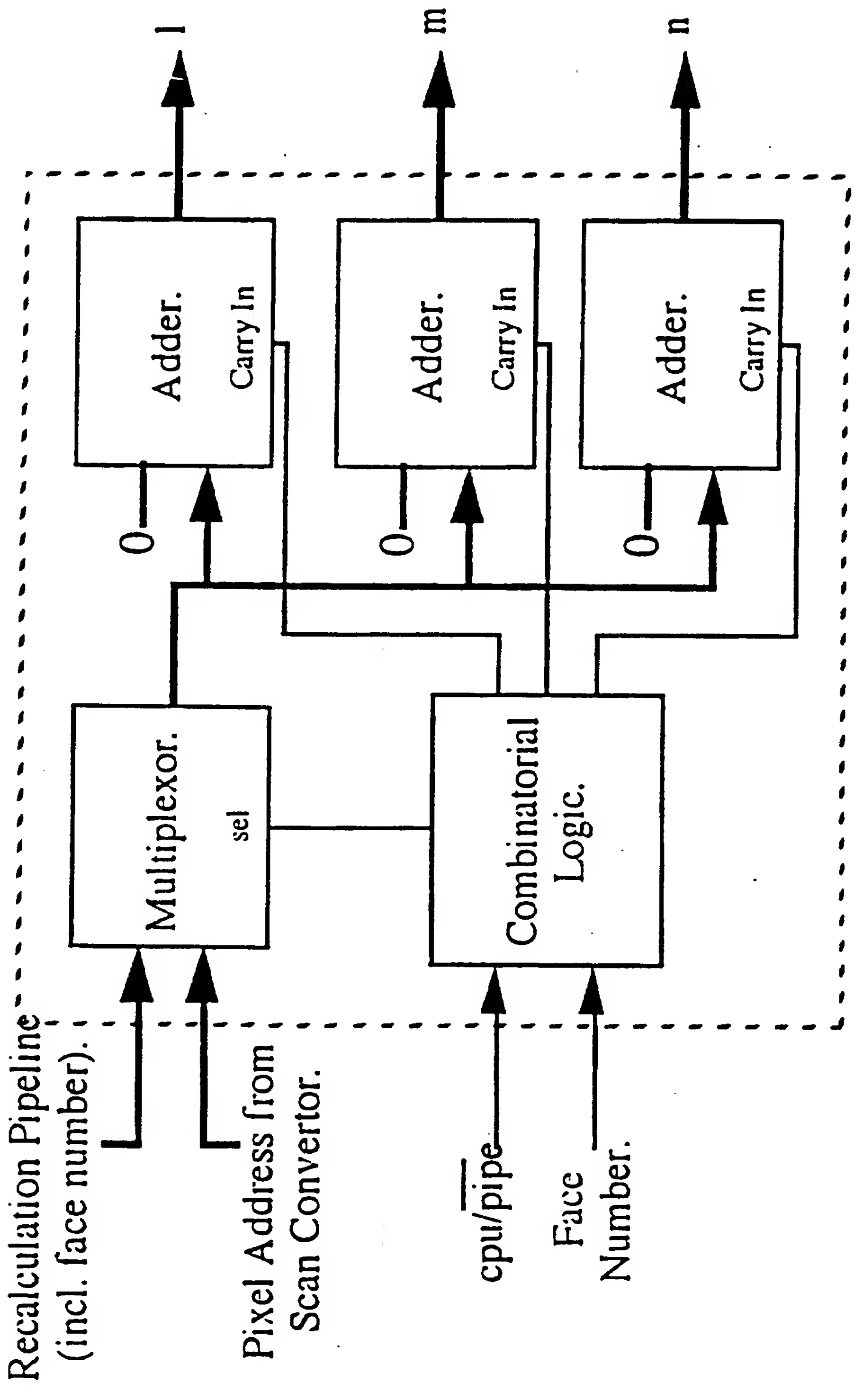

l
m
n
Adder.
Carry In
Adder.
Carry In
Adder.
Carry In
0
0
0
Multiplexor.
sel
Combinatorial Logic.
Recalculation Pipeline (incl. face number).
Pixel Address from Scan Convertor.
$\overline{\text{cpu}}$/pipe
Face Number.

FIGURE 24

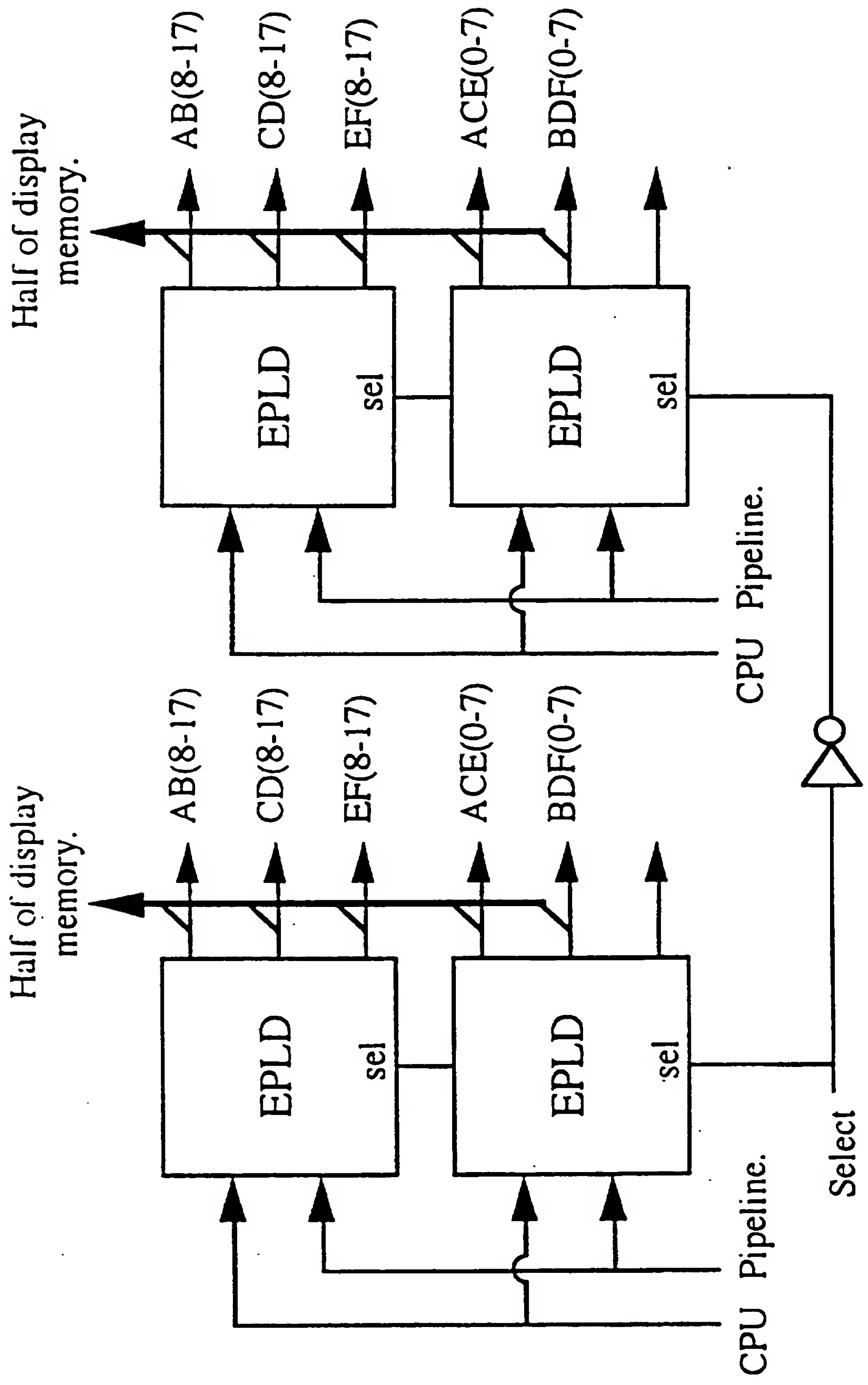

Half of display memory.
AB(8-17)
CD(8-17)
EF(8-17)
ACE(0-7)
BDF(0-7)
EPLD
sel
EPLD
sel
CPU
Pipeline.
Half of display memory.
AB(8-17)
CD(8-17)
EF(8-17)
ACE(0-7)
BDF(0-7)
EPLD
sel
EPLD
sel
CPU
Pipeline.
Select

FIGURE 25

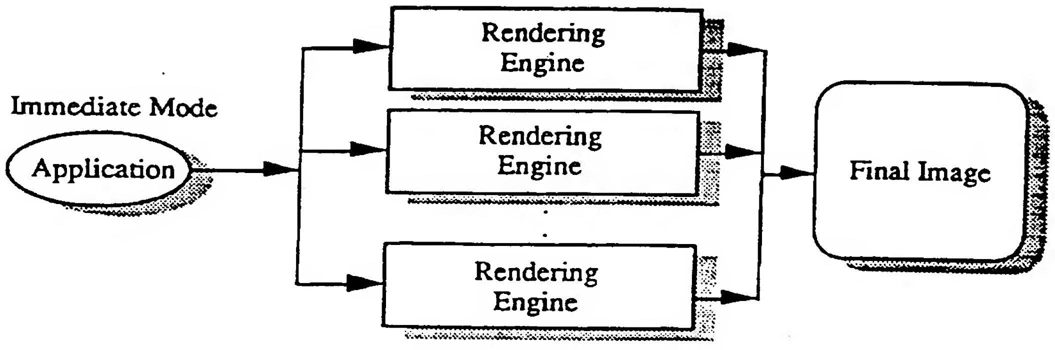


FIGURE 26

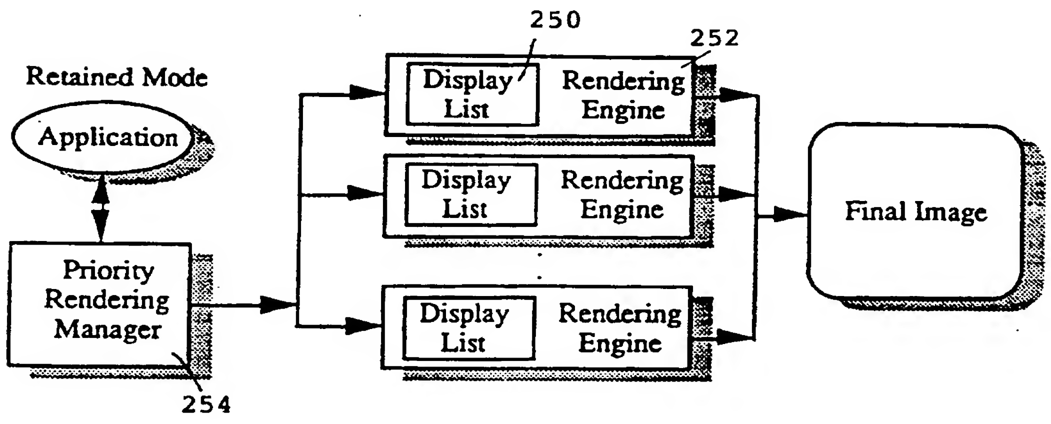


FIGURE 27

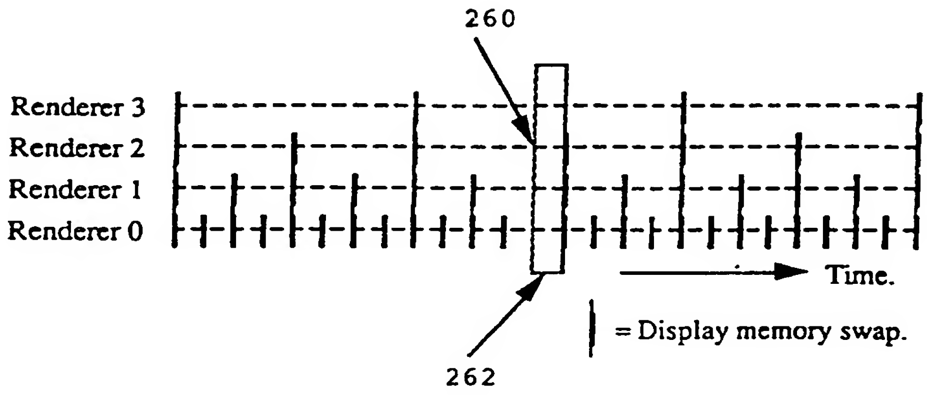


FIGURE 28

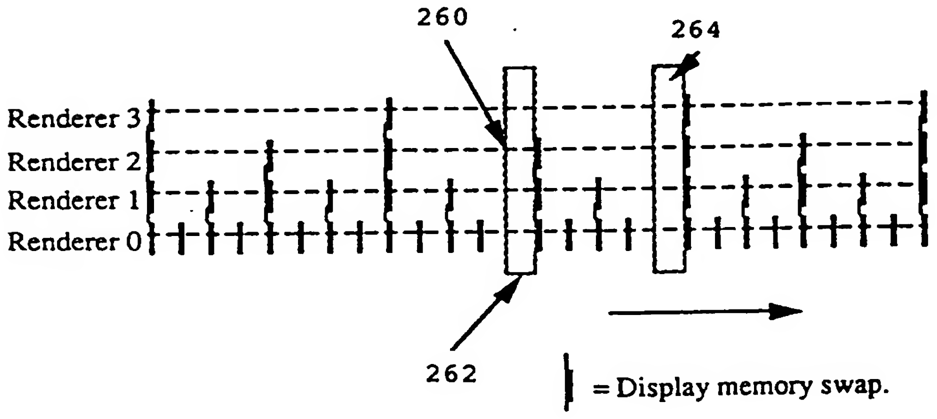


FIGURE 29

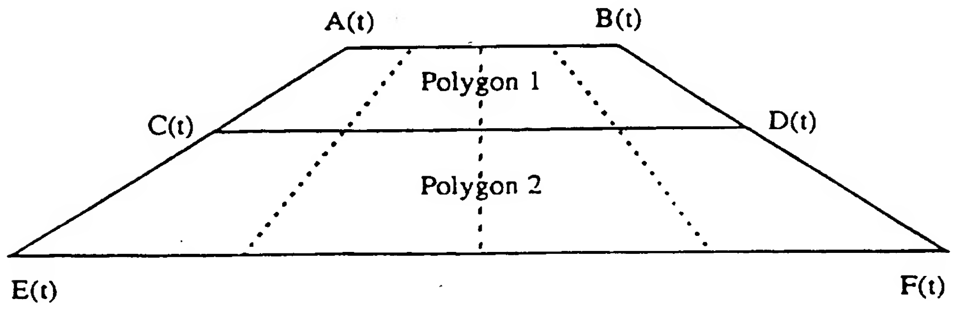


FIGURE 30

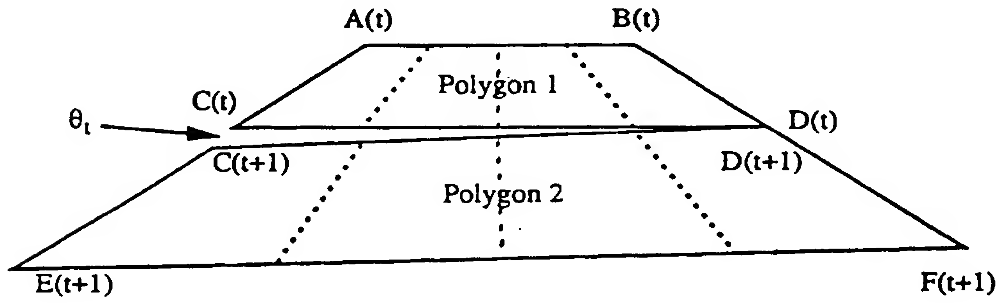


FIGURE 31

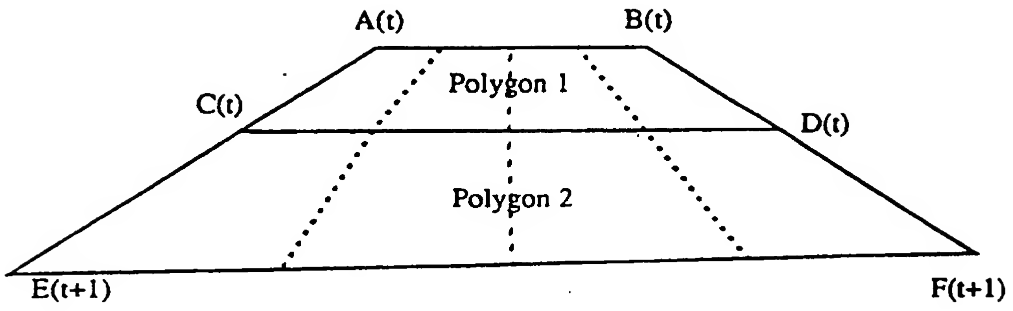


FIGURE 32

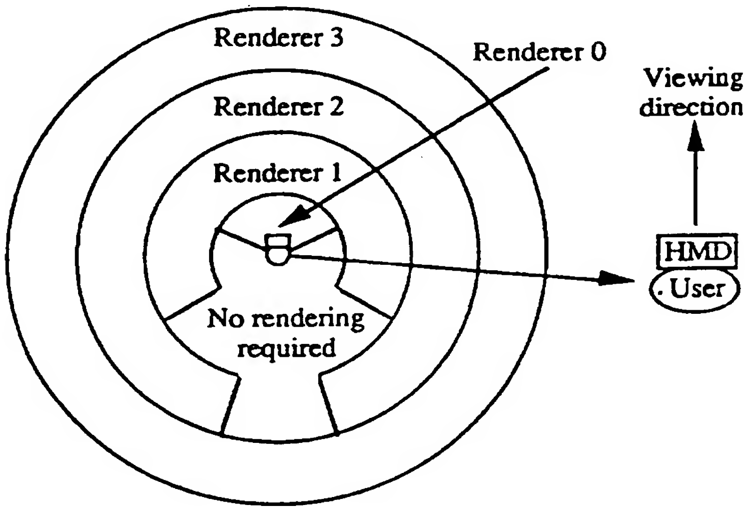


FIGURE 33

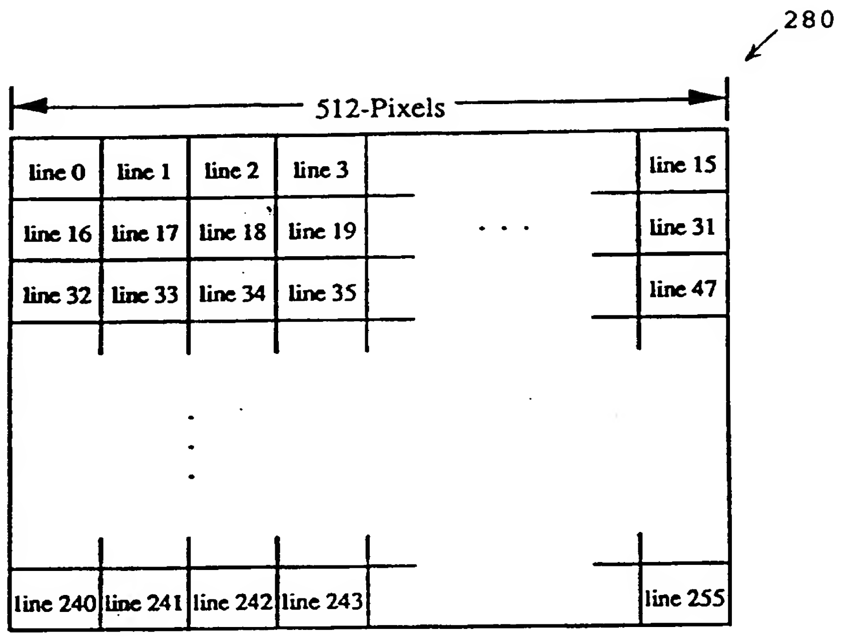

280
512-Pixels
line 0
line 1
line 2
line 3
line 15
line 16
line 17
line 18
line 19
line 31
line 32
line 33
line 34
line 35
line 47
line 240
line 241
line 242
line 243
line 255

FIGURE 34

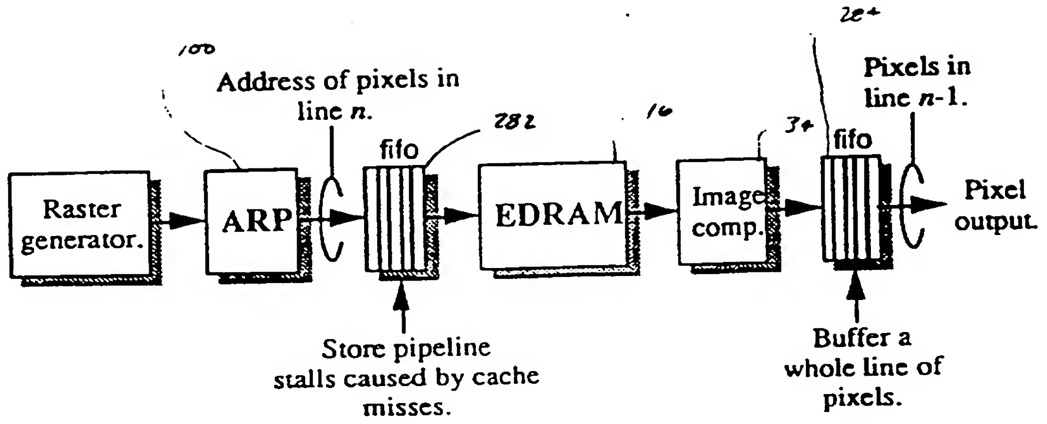

100
Address of pixels in line *n*.
fifo
282
16
34
284
fifo
Pixels in line *n*-1.
Raster generator.
ARP
EDRAM
Image comp.
Pixel output.
Store pipeline stalls caused by cache misses.
Buffer a whole line of pixels.

FIGURE 35

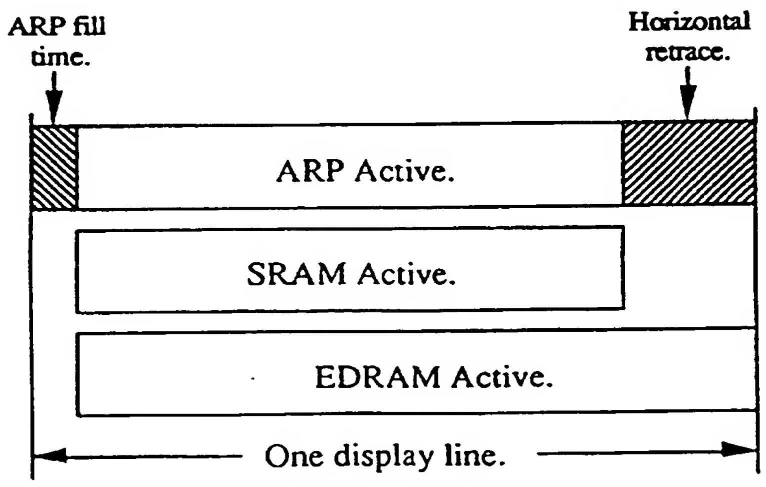


FIGURE 36

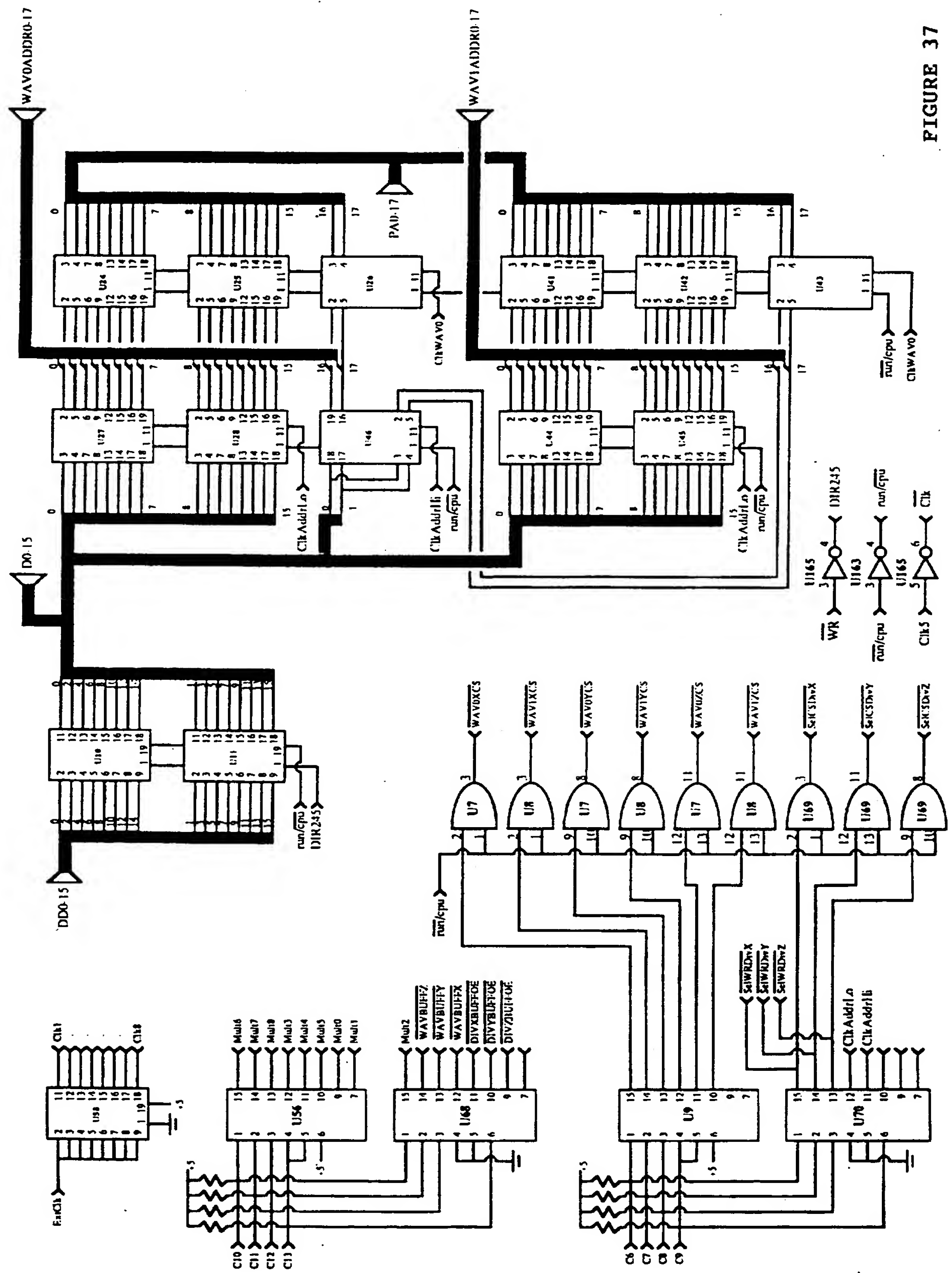

WAV0ADDR0-17
WAV1ADDR0-17
PAD-17
D0-15
DD0-15
ClkWAV0
run/cpu
ClkAddrLo
ClkAddrHi
DIR245
WR
Clk
Clk5
U165
U163
WAV0XCS
WAV1XCS
WAV0YCS
WAV1YCS
WAV0ZCS
WAV1ZCS
SelCSDivX
SelCSDivY
SelCSDivZ
U7
U8
U169
SelWRDivX
SelWRDivY
SelWRDivZ
WAVBUFFZ
WAVBUFFY
WAVBUFFX
DIVXBUFFOE
DIVYBUFFOE
DIVZBUFFOE
U9
U70
U56
U68
U58
ExtClk
+5

FIGURE 37

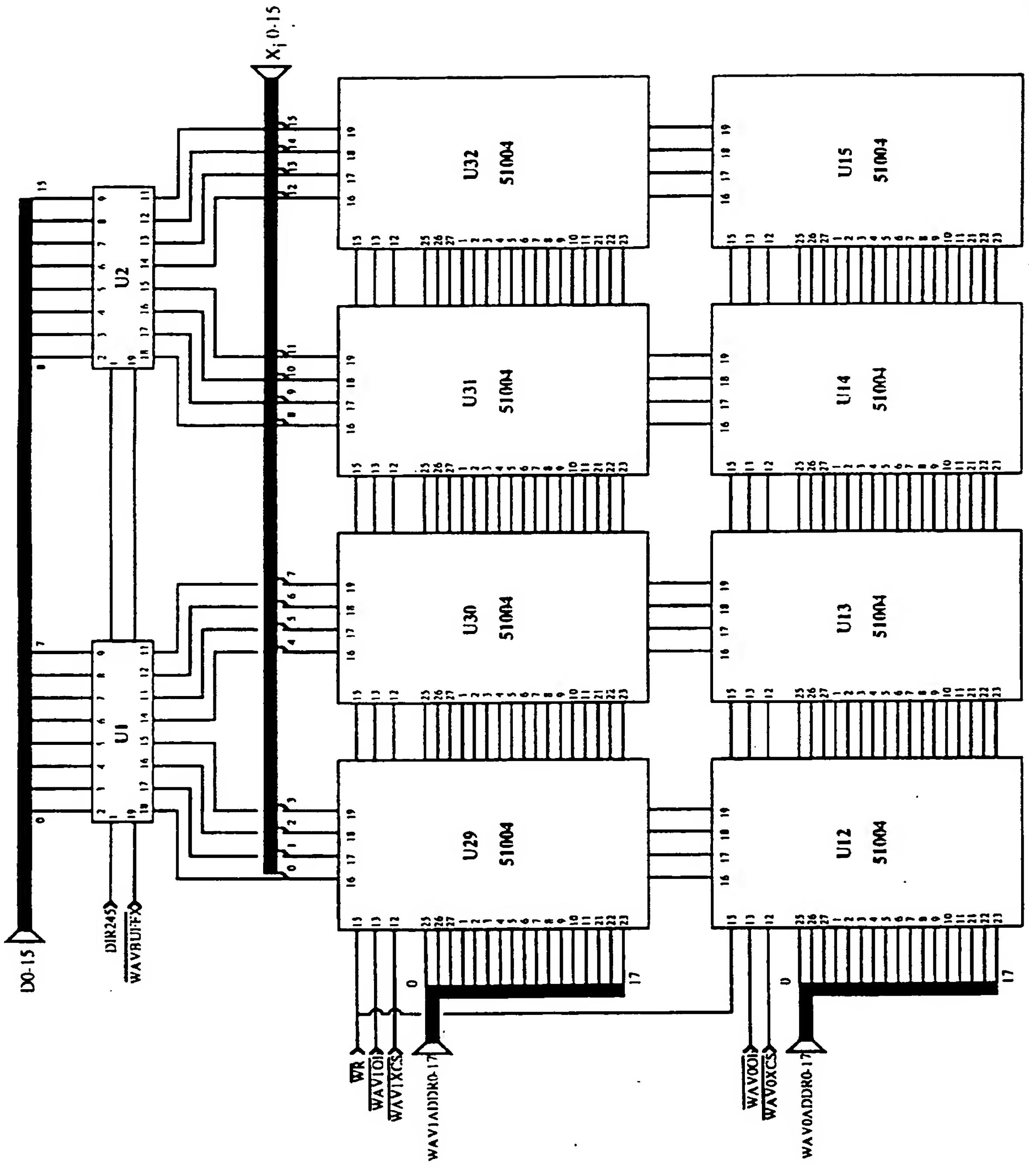

D0-15
DIR245
WAVBUFFX
X_i 0-15
U1
U2
U29 51004
U30 51004
U31 51004
U32 51004
U12 51004
U13 51004
U14 51004
U15 51004
WR
WAV1OE
WAV1XCS
WAV1ADDR0-17
WAV0OE
WAV0XCS
WAV0ADDR0-17

FIGURE 38

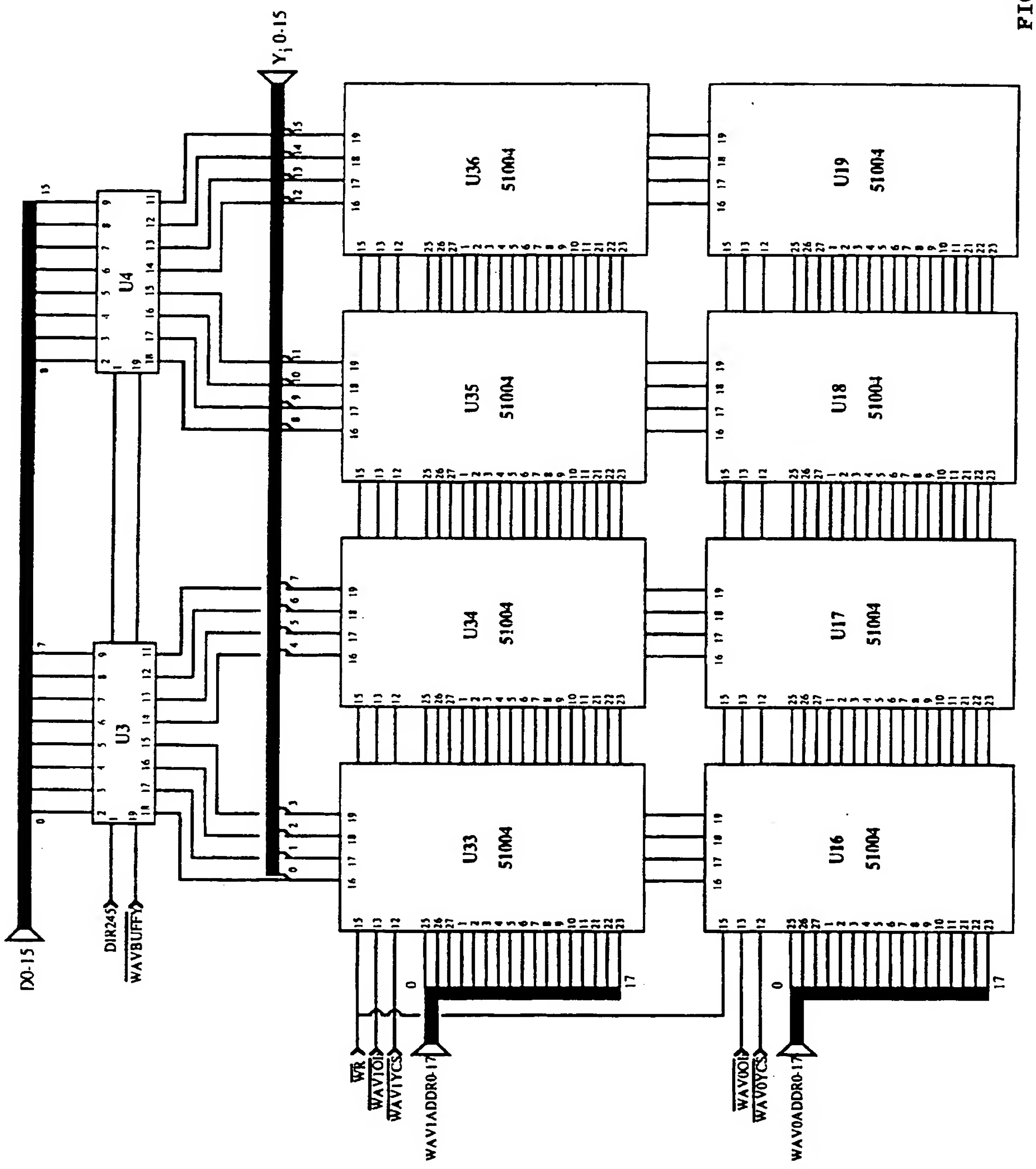
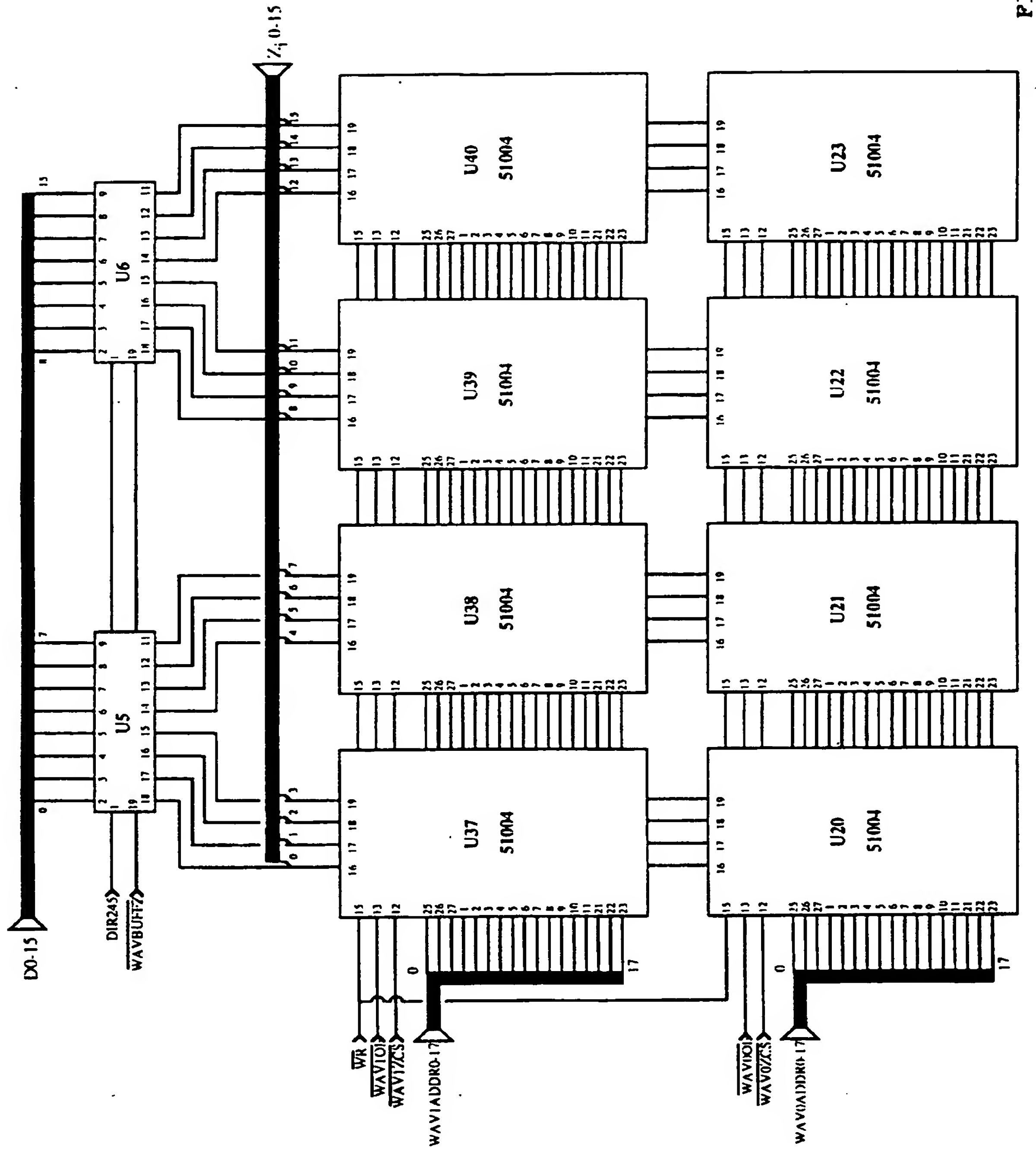

Y_j 0-15
U36
51004
U19
51004
U35
51004
U18
51004
U34
51004
U17
51004
U33
51004
U16
51004
U4
U3
D0-15
DIR245
WAVBUFF
WR
WAV1OE
WAV1CS
WAV1ADDR0-17
WAV0OE
WAV0CS
WAV0ADDR0-17

FIGURE 39

FIGURE 40



SUBSTITUTE SHEET (Rule 26)

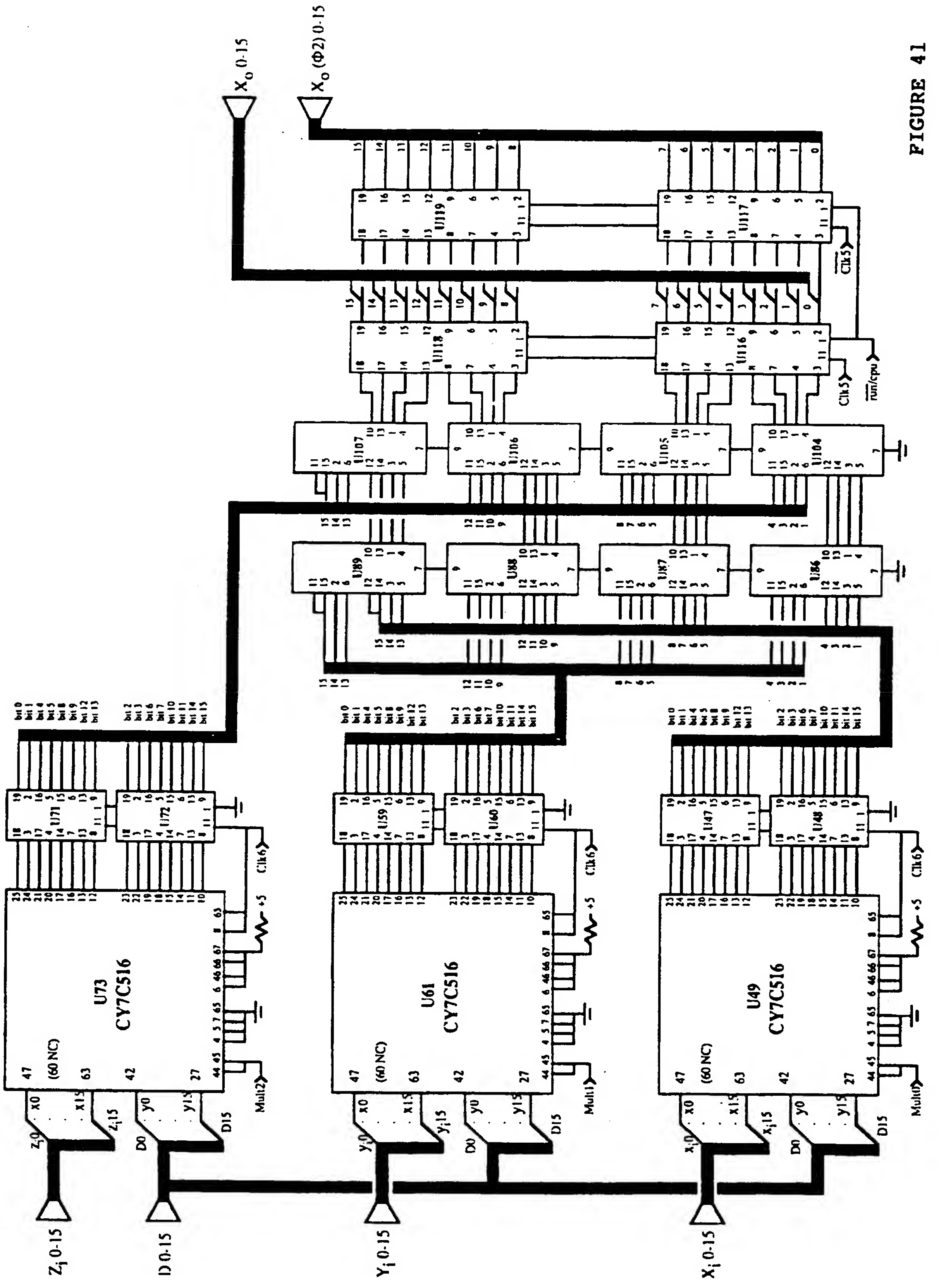

X_o 0-15
X_o (Φ2) 0-15
U119
U117
U118
U116
Clk5
run/cpu
U107
U106
U105
U104
U89
U88
U87
U86
U71
U72
U59
U60
U47
U48
Clk6
+5
U73
CY7C516
U61
CY7C516
U49
CY7C516
(60 NC)
Mult2
Mult1
D15
Z_i 0-15
D 0-15
Y_i 0-15
X_i 0-15

FIGURE 41

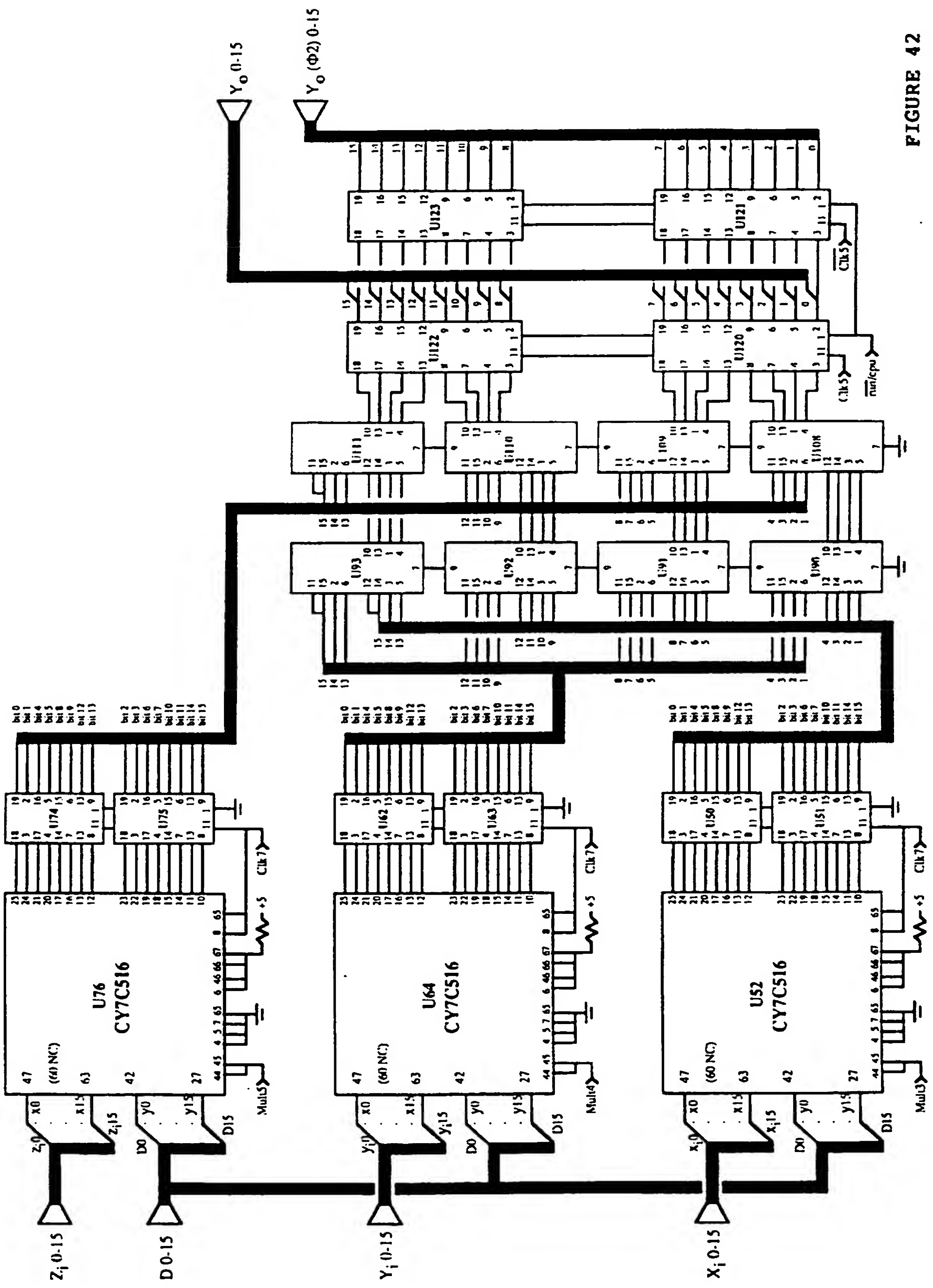
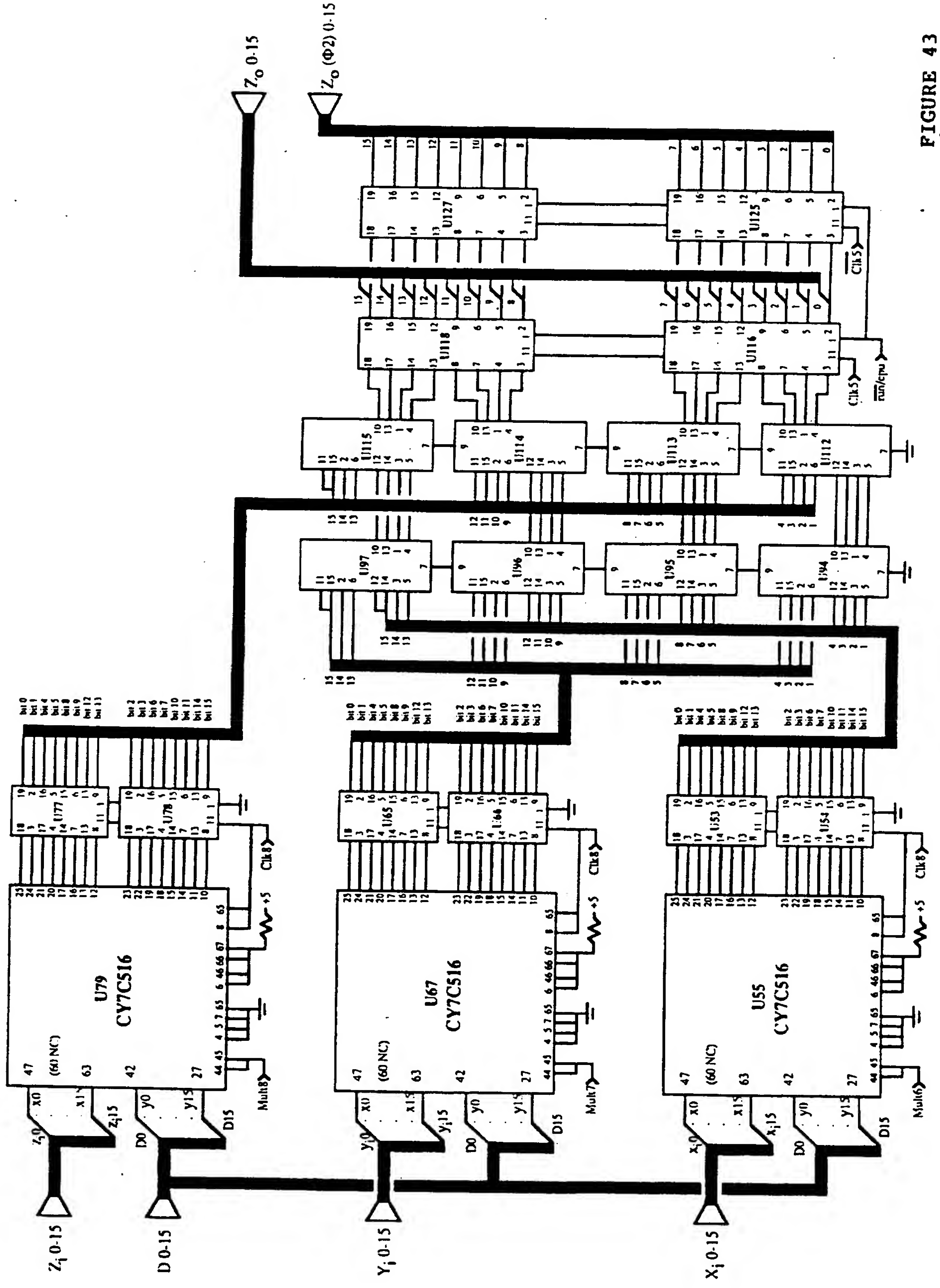
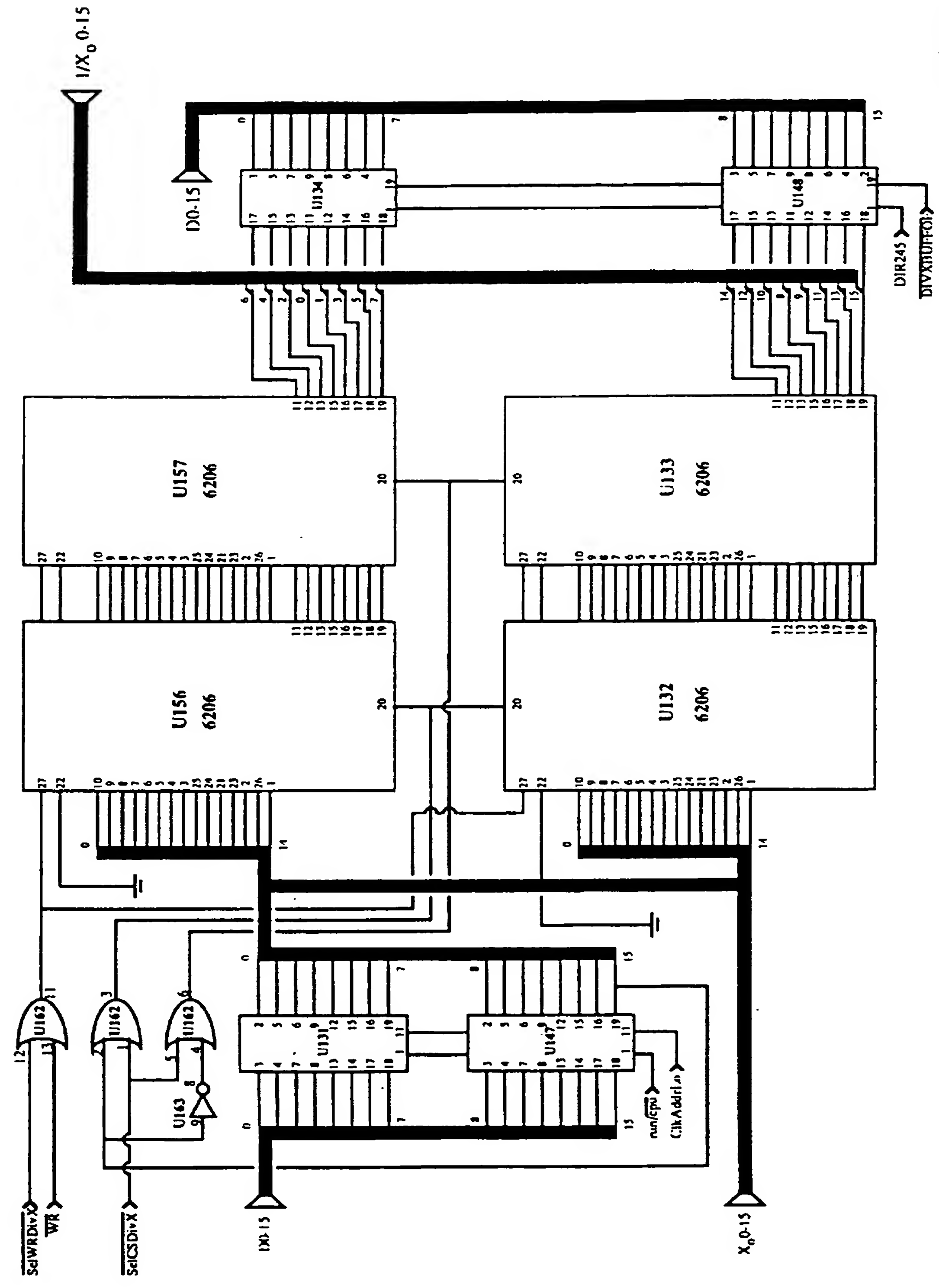

Y_o 0-15
Y_o (Φ2) 0-15
U123
U121
U122
U120
U111
U110
U109
U108
U93
U92
U91
U90
Clk5
run/cpu
U174
U175
U162
U163
U150
U151
Clk7
+5
U76 CY7C516
U64 CY7C516
U52 CY7C516
(60 NC)
Mult5
Mult4
Mult3
Z_i 0-15
D 0-15
Y_i 0-15
X_i 0-15
x0
x15
y0
y15
D0
D15

FIGURE 42



Z_o 0-15
Z_o (Φ2) 0-15
U127
U125
U118
U116
U115
U114
U113
U112
U97
U96
U95
U94
U77
U78
U65
U66
U53
U54
U79 CY7C516
U67 CY7C516
U55 CY7C516
Clk5
run/cpu
Clk8
Mult8
Mult7
Mult6
+5
Z_i 0-15
D 0-15
Y_i 0-15
X_i 0-15

FIGURE 43



1/X$_0$ 0-15
D0-15
U134
U148
DIR245
DIVXBUFFOE
U157 6206
U133 6206
U156 6206
U132 6206
U162
U163
U131
U147
run/cpu
ClkAddrL
SelWRDivX
WR
SelCSDivX
D0-15
X$_0$0-15

FIGURE 44

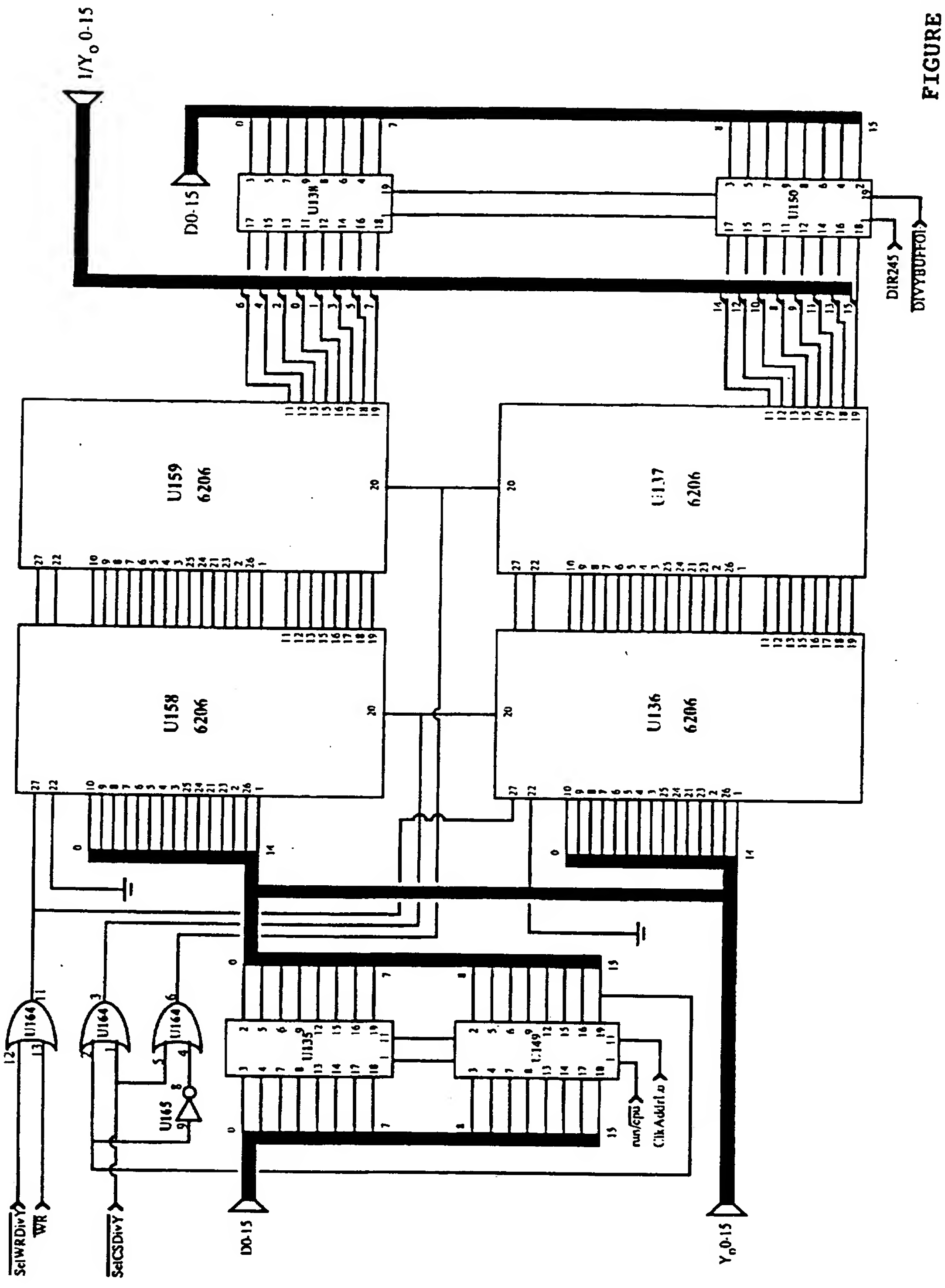
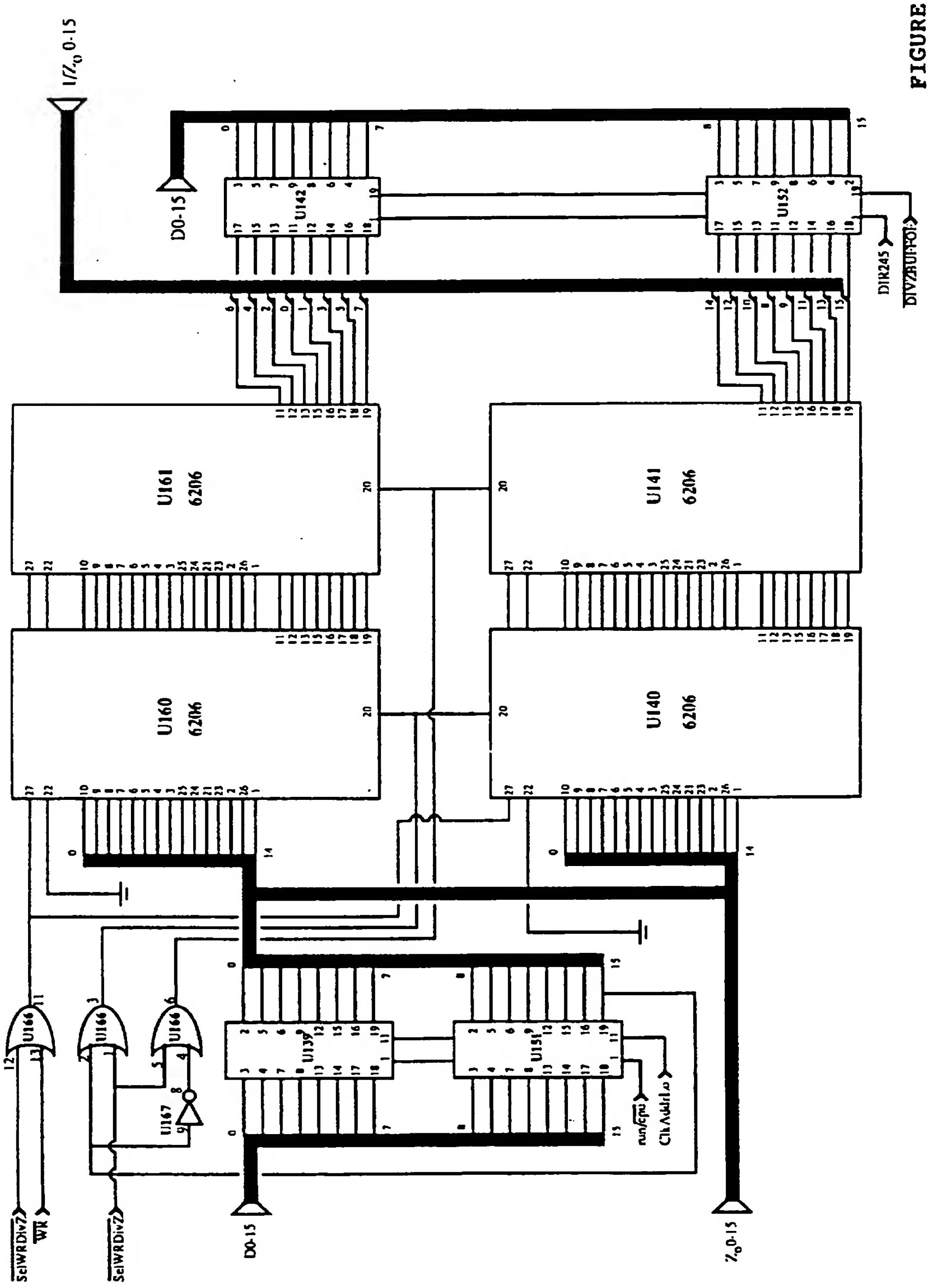

1/Y0 0-15
D0-15
U138
U150
DIR245
DIVYBUFFOE
U159 6206
U137 6206
U158 6206
U136 6206
U164
U165
U135
U149
run/cpu
ClkAddrLo
SelWRDivY
WR
SelCSDivY
D0-15
Y0 0-15

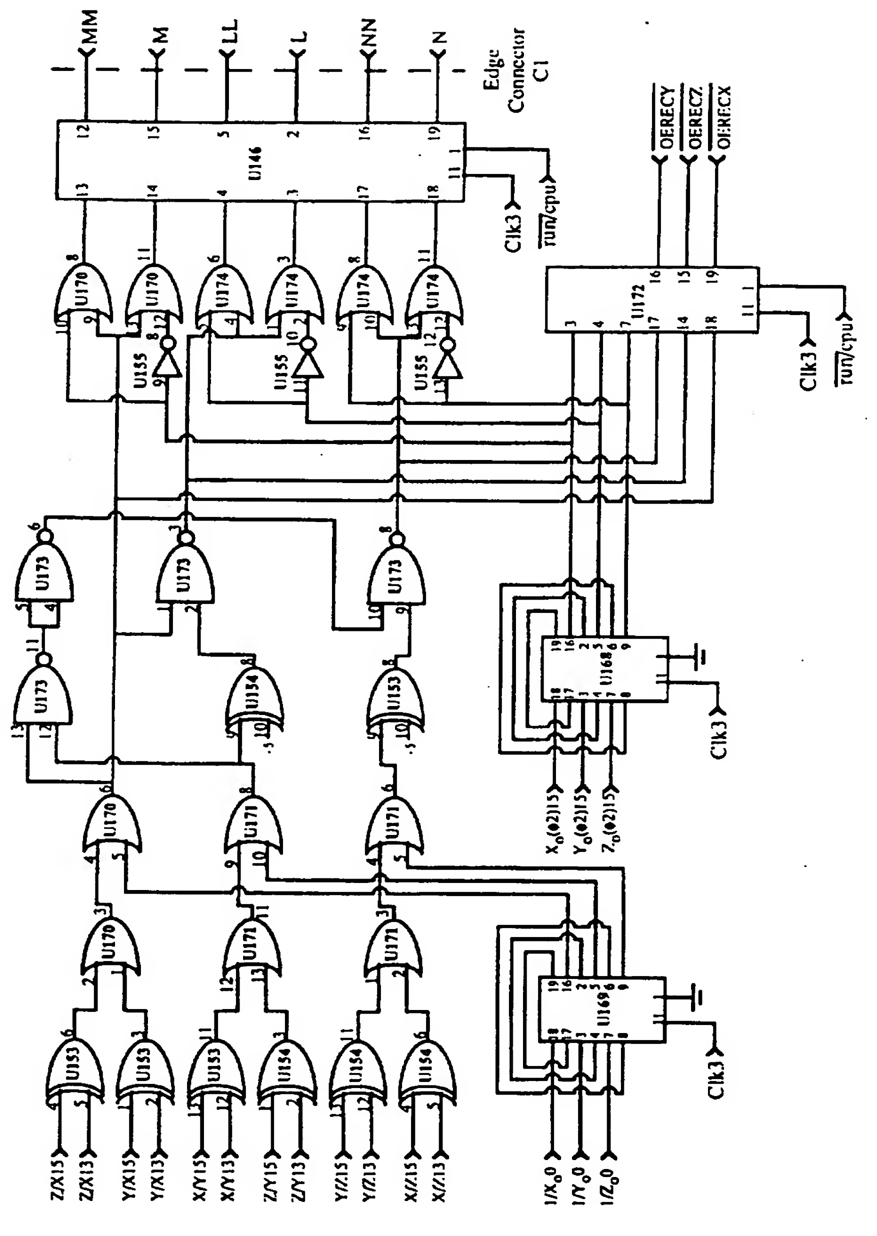
FIGURE 45



I/Z0 0-15
D0-15
U142
U152
DIR245
DIV2BUFFOE
U161
6206
U141
6206
U160
6206
U140
6206
U166
U167
U139
U151
SelWRDiv7
WR
SelWRDiv7
D0-15
Z0 0-15
run/cpu
ClkAddrLo

FIGURE 46

FIGURE 47


MM
M
LL
L
NN
N
Edge Connector C1
U146
Clk3
run/cpu
OERECY
OERECZ
OERECX
U1172
U170
U174
U155
U173
U153
U154
U171
U168
U169
Xo(⊘2)15
Yo(⊘2)15
Zo(⊘2)15
1/Xo0
1/Yo0
1/Zo0
Z/X15
Z/X13
Y/X15
Y/X13
X/Y15
X/Y13
Z/Y15
Z/Y13
Y/Z15
Y/Z13
X/Z15
X/Z13

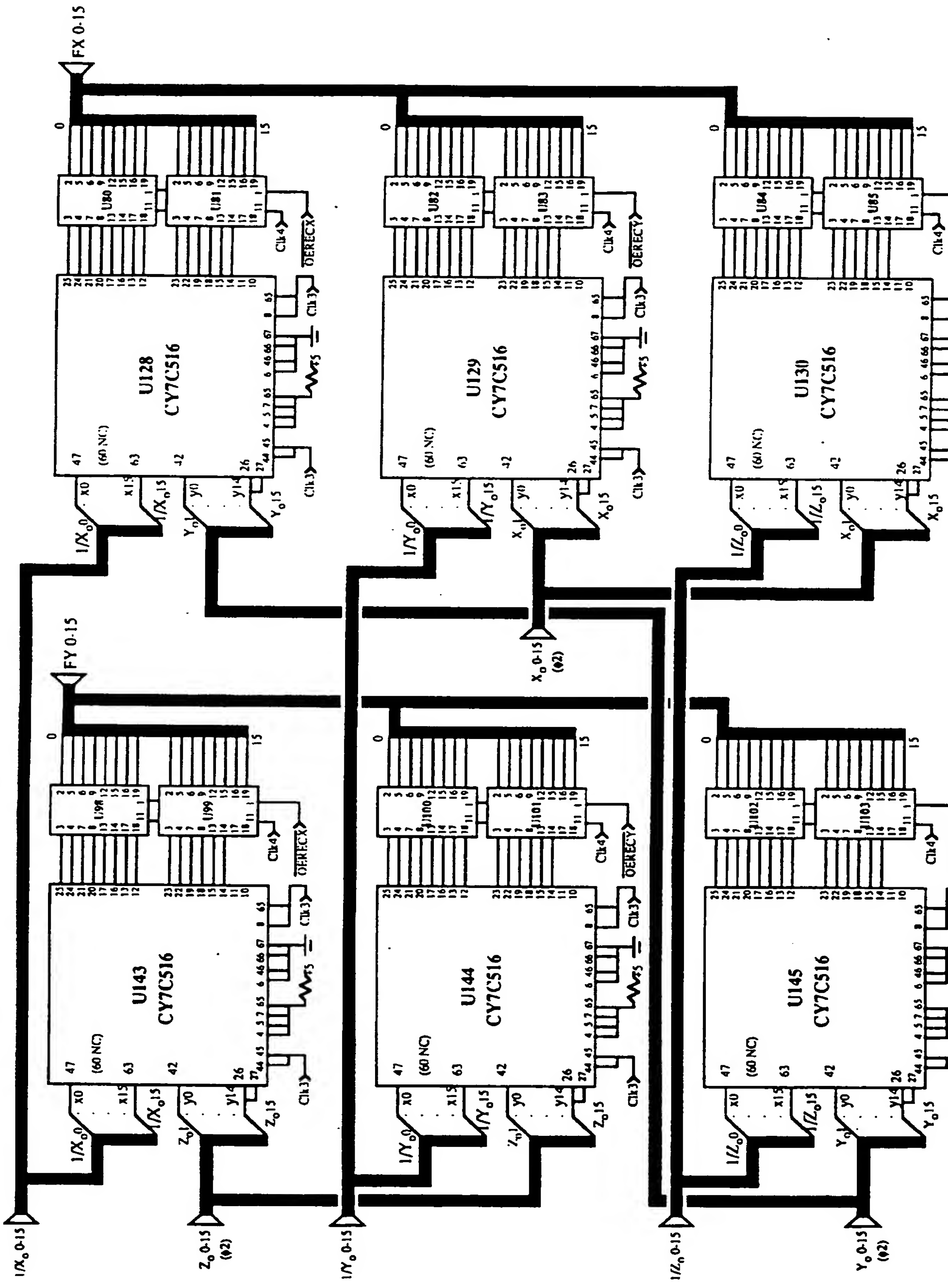

FX 0-15
FY 0-15
U128
CY7C516
U129
CY7C516
U130
CY7C516
U143
CY7C516
U144
CY7C516
U145
CY7C516
U80
U81
U82
U83
U84
U85
U98
U99
U100
U101
U102
U103
Clk4
Clk3
OERECX
OERECY
OERECZ
X_0 0-15 (#2)
$1/X_0$ 0-15
Z_0 0-15 (#2)
$1/Y_0$ 0-15
$1/Z_0$ 0-15
Y_0 0-15 (#2)

FIGURE 48

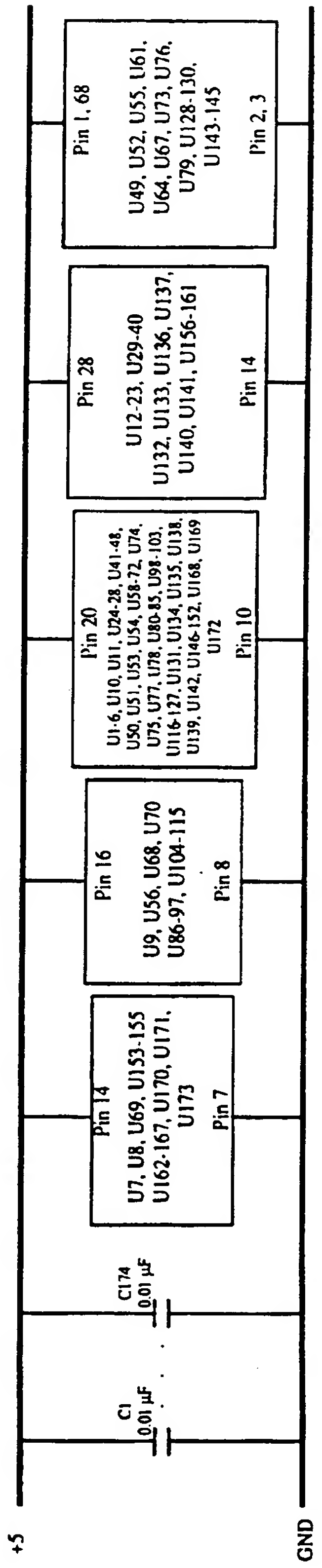

+5
GND
C1
0.01 μF
C174
0.01 μF
Pin 14
U7, U8, U69, U153-155
U162-167, U170, U171,
U173
Pin 7
Pin 16
U9, U56, U68, U70
U86-97, U104-115
Pin 8
Pin 20
U1-6, U10, U11, U24-28, U41-48,
U50, U51, U53, U54, U58-72, U74,
U75, U77, U78, U80-85, U98-103,
U116-127, U131, U134, U135, U138,
U139, U142, U146-152, U168, U169
U172
Pin 10
Pin 28
U12-23, U29-40
U132, U133, U136, U137,
U140, U141, U156-161
Pin 14
Pin 1, 68
U49, U52, U55, U61,
U64, U67, U73, U76,
U79, U128-130,
U143-145
Pin 2, 3

FIGURE 49

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 95/00445

A. CLASSIFICATION OF SUBJECT MATTER

Int Cl^6: G06T 1/60, 15/00 GO6F 12/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06T 1/60, 15/00; GO6F 12/00,12/02,12/06,12/08,12/10,15/62,15/72

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
AU: IPC as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
DERWENT AND JAPIO:} ACCESS: MEMORY ADDRESS IMAGE DISPLAY: PIXEL: RENDER: VIRTUAL REALITY ROTATION: ORIENTATE:

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US, a 4477802 (Walter et al) 16 October 1984 See line 1 col 3- line 39 col 10, Figs 1-4.	1,10,25-28,30,57
X	Patent Abstracts of Japan, E1600,page 51,JP,A 6-153026 (VICTOR CO OF JAPAN) 31 May 1994	1,10,25-28,30,57

[X] Further documents are listed in the continuation of Box C

[X] See patent family annex

*	Special categories of cited documents:
"A"	document defining the general state of the art which is not considered to be of particular relevance
"E"	earlier document but published on or after the international filing date
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O"	document referring to an oral disclosure, use, exhibition or other means
"P"	document published prior to the international filing date but later than the priority date claimed
"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&"	document member of the same patent family

Date of the actual completion of the international search
27 OCTOBER 1995

Date of mailing of the international search report
6 NOVEMBER 1995

Name and mailing address of the ISA/AU
AUSTRALIAN INDUSTRIAL PROPERTY ORGANISATION
PO BOX 200
WODEN ACT 2606
AUSTRALIA Facsimile No.: (06) 285 3929

Authorized officer
John Thomson
JOHN THOMSON
Telephone No.: (06) 283 2214

C (Continuation)	DOCUMENTS CONSIDERED TO BE RELEVANT	
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Patent Abstracts of Japan, P1414, page 44, JP, A 4-141784 (OMRON CORP) 15 May 1992	1,10,25-28,30,57
X	Patent Abstracts of Japan,P988, page 55, JP, A 1-259398 (SEGA ENTERP K.K.) 17 October 1989	1,10,25-28,30,57
X	Patent Abstracts of Japan, P557, page 95, JP, A 61-240382 (MATSUSHITA ELECTRIC IND CO LTD) 25 October 1986	1,10,25-28,30,57
X	Patent Abstracts of Japan, P519, page 125, JP, A 61-148578,(PHOTO COMPOSING MACH MFG CO LTD) 7 July 1986	1,10,25-28,30,57
X	Patent Abstracts of Japan, P436, page 124 JP, A 60-205674, (FUJITSU K.K.) 17 October 1985	1,10,25-28,30,57
A,P	US, A 5426733 (MASUI) 20 June 1995	
A	EP, A 502643 (FUJITSU LIMITED) 9 September 1992	

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 95/00445

Box I **Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)**

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a)

Box II **Observations where unity of invention is lacking (Continuation of item 2 of first sheet)**

This International Searching Authority found multiple inventions in this international application, as follows:

1) A method and apparatus for selecting and displaying image pixel data for an image by addressing stored image data on the basis of a determined orientation of the image to be displayed as in claims 1 - 60.
2) A graphics display system with a pluarality of memories for storing image data based on characteristics of the image data and selecting memories for display data acording to characteristics of the image to be displayed as in claims 61-90.

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims

2. [X] As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest ☐ The additional search fees were accompanied by the applicant's protest.
☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT
Information on patent family members

International Application No.
PCT/AU 95/00445

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member					
US	4477802	DE	3279548	EP	82746	IL	67374
US	5426733	JP	6044368	US	5384645	JP	5108812
EP	502643	JP	4277871	US	5442734		

END OF ANNEX